Volumen 1
Fundamentos de Linux

Unidad 1: Fundamentos de Linux

Objetivos de Aprendizaje

Al finalizar esta unidad, Ud será capaz de:

- Explicar que es un sistema operativo
- Explicar las características de un sistema operativo UNIX
- Discutir la evolución de Linux
- Explicar los requerimientos de hardware de Linux
- Discutir acerca del kernel (núcleo), el shell (interprete de comandos) y el sistema de archivos en Linux
- Discutir la capacidad de Linux para procesar texto, programas y proporcionar documentación de los comandos.

1. Introducción

Un sistema de computadora es un dispositivo electrónico programable que puede almacenar, recuperar y procesar datos. Ejecuta un conjunto de instrucciones llamado programa. La ejecución del programa da al usuario final el resultado deseado. Un ejemplo de ejecución de un programa es cuando se trabaja con una calculadora en un sistema operativo Windows.

Una computadora es capaz de realizar múltiples tareas, tales como:

- Ejecutar programas de usuario
- Conectar computadoras en una Red de Área Local (Local Área Network – LAN)
- Compartir los recursos
- Controlar su hardware

Para que una computadora sea capaz de hacer todo esto, requiere de un programa especial llamado *sistema operativo.* El sistema operativo reside en el disco duro de la computadora y actúa como un puente entre los programas de usuario y los programas que controlan le hardware de la computadora.

El sistema operativo trabaja principalmente en segundo plano. Se encarga de los componentes hardware de una computadora, además de asegurar el inicio y ejecución de diversos programas. También se encarga de los datos almacenados en los dispositivos de almacenamiento de la computadora.

El sistema operativo es el responsable de las siguientes funciones del sistema de computadora:

- Arrancar o iniciar la computadora
- Actuar como interfaz entre el CPU y el mundo externo
- Coordinar los dispositivos del sistema
- Coordinar las aplicaciones o programas en ejecución

El sistema operativo controla todo el trabajo de la computadora. Sin el, la computadora solo es un montón de circuitos electrónicos.

La figura 1.1 ilustra el rol de un sistema operativo

Figura 1.1 Rol del Sistema Operativo

Los siguientes son algunos de los usos del sistema operativo:

- Proporciona diferentes recursos a los usuarios, tales como: calculo, almacenamiento, dispositivos de Entrada/Salida (Input/Output – I/O) manejo de red, etc
- Permite que varios usuarios trabajen juntos compartiendo e intercambiando programas, aplicaciones y datos en la misma instalación.
- Ayuda a resolver conflictos cuando los usuarios solicitan el mismo recurso simultáneamente.
- Proporciona seguridad cuando los usuarios comparten datos y programas.
- Asiste en la administración y evaluacion del uso y eficacia de un sistema, recolectando datos sobre la utilización de los recursos.
- En los sistemas de computadoras, existen varios componentes que solo trabajan en forma secuencial y que solo se pueden compartir directamente en forma marginal. Los sistemas operativos permiten que se haga un uso eficiente de estos

componentes.

Inicio de un Sistema Operativo

- El proceso de iniciar el sistema operativo es llamado *arranque* (bootstrapping o booting). Aquí se mencionan algunos puntos acerca de cómo se inicia un sistema operativo.
- Las instrucciones para el arranque están incluidas en uno de los chips de la computadora, llamado el chip BIOS (Basic Input/Output System)
- El chip BIOS informa a la computadora que busque un programa especial llamado el gestor de arranque (boot loader)
- El gestor de arranque esta disponible en un lugar fijo en el disco de arranque. El disco de arranque en cualquier computadora es el disco duro primario.
- El gestor de arranque inicia la parte principal del sistema operativo.

Los sistemas operativos son clasificados en dos tipos:

- **Sistema Operativo de Usuario Único:** permite que solo un usuario use todos los recursos del sistema de computadora en cualquier momento dado. Mientras el procesador maneja un usuario o programa, otros programas o usuarios no pueden usar estos recursos. Un ejemplo de este tipo el sistema operativo DOS.
- **Sistema Operativo Multiusuario:** permite que más de un usuario o programa se ejecuten o usen recursos del sistema en forma simultánea. Algunos ejemplos de estos sistemas operativos multiusuario son UNIX, XENIX, Linux, Solaris, etc.

En los tiempos en que los sistemas operativos como DOS únicamente permitían solo un usuario para realizar tareas simples, la introducción de UNIX creo una nueva forma de computación. A pesar que Windows también es un sistema operativo principalmente de usuario único, permite que varias tareas se realicen en forma simultanea. A continuación se aprenderá acerca del sistema operativo UNIX, dado que Linux es una variación del sistema operativo UNIX.

2. Sistema Operativo UNIX

El sistema operativo UNIX tiene una historia muy interesante. Algunos aspectos de su evolución son:

- Bell Telephone Laboratorios (BTL), GE y MIT se unieron intentando crear un sistema operativo que permitiera trabajar simultáneamente hasta mil usuarios. Ken Thompson y Dennis Ritchie de BTL trabajaron en esto y crearon un sistema operativo llamado MULTICS, un acrónimo de MULTiplexed Information and Computer Services (Servicio de Información Multiplexada y Calculo)
- BTL se retiro del proyecto durante el desarrollo del trabajo.
- En 1969-70, Thompson y Ritchie reescribieron el sistema operativo para jugar un juego de guerra espacial con otra maquina mas pequeña.
- Este sistema operativo fue llamado Uniplexed Information and Computing Services (UNICS), un juego de palabras del MULTICS original.
- El nombre de UNICS luego fue transformado a UNIX.
- En 1972-73, el sistema UNIX fue reescrito usando el lenguaje de programación C
- El sistema operativo UNIX es uno de los mas poderosos versátiles y flexibles en el mundo hoy en día.

El sistema operativo UNIX corre en un rango de computadores desde microcomputadoras hasta mainframes. Algunas de sus características son:

- Es conocido como un sistema operativo **abierto** dado que puede ser llevado e instalado en cualquier clase de sistema de computadora y plataforma de hardware.
- Normalmente, los sistemas operativos son escritos en lenguaje ensamblador. Sin embargo, UNIX esta escrito en un lenguaje de alto nivel y su código fuente esta disponible fácilmente. Esto supone una ventaja para los programadores cuando incorporan cambios para ajustarse a sus necesidades.

- Es un sistema multiusuario y multitarea. Multitarea significa que el sistema operativo coordina múltiples tareas o trabajos en forma simultanea. Mientras un usuario esta compilando un programa en C, otro puede crear documentos usando un editor, cada ignorando la presencia del otro.
- UNIX es uno de los sistemas operativos más poderosos existentes, por el hecho de poder compartir recursos en tiempo real.
- A pesar de que UNIX esta desarrollado para programadores, proporciona un entorno tan flexible que también es usado en negocios, ciencias, educación e industria.
- Los interruptores de telecomunicación y sistemas de transmisión son algunos de ejemplos del uso del sistema operativo UNIX

A continuación se aprenderá acerca del sistema operativo Linux

3. Sistema Operativo Linux

Linux es un sistema operativo distribuido gratuitamente basado en el sistema operativo UNIX. Fue desarrollado originalmente por Linus Torvalds, quien empezó a trabajar sobre Linux en 1991 siendo estudiante de la Universidad de Helsinki en Finlandia. Luego, miles de programadores contribuyeron a su desarrollo y fue distribuido gratuitamente por Internet.

Por los años 80, los sistemas operativos eran básicamente propietarios, lo que significaba que se tenía que usar solo el sistema operativo proporcionado para una plataforma especifica.

El proyecto GNU fue fundado por Richard Stallman, quien fue el también fundador de Free Software Foundation (FSF), autor de GNU GPL (General Public License) y el desarrollador original de algunos programas de software GNU (por ejemplo, el compilador gcc y el editor de texto Emacs)

Las principales metas del proyecto GNU incluyeron las siguientes:

- Desarrollar un sistema operativo compatible con UNIX
- Soportar diferentes arquitecturas de hardware
- Hacer que el sistema operativo estuviese disponible libre de costo para asegurar que los usuarios pudiesen redistribuir todo el sistema y cambiar o contribuir a alguna parte de el.

En 1990, la mayoría de las piezas de software del sistema operativo basado en GNU se habían escrito, excepto la más importante, el kernel. El kernel es el núcleo del sistema operativo.

Mas tarde, el kernel gratuito basado en UNIX, desarrollado por Linus Torvalds fue combinado con el sistema GNU. Así nació un sistema operativo, el sistema GNU basado en Linux.

Las etapas significativas en la evolución de Linux son:

- En 1991, Linus Torvalds desarrollo Linux con el soporte de desarrolladores a lo largo del mundo y lo llamo Linux
- El lanzo la versión 0.02 de Linux en 1991
- En 1994, fue lanzada la versión 1.0 de Linux
- La versión 2.6 actual, completa fue lanzada en Diciembre de 2003. Sin embargo, su desarrollo continúa.

Los siguientes son algunos de los hechos básicos acerca del sistema operativo Linux:

- Es desarrollado, escrito, distribuido y respaldado bajo GPL de GNU (GNU no es UNIX). Como resultado, su código fuente puede ser distribuido gratuitamente y disponible para el publico en general.
- Los sistemas Linux se usan para redes, desarrollo de software, soluciones de alojamiento basados en Web y como plataforma de usuario final.
- La mascota oficial, que Linus eligió para su sistema operativo, es el pingüino Linux

llamado Tux (Torvalds Unix), que se presenta en la figura 1.2

Figura 1.2: Tux la mascota de Linux

- Linux no es un derivado del código fuente de UNIX. Sin embargo, la interfaz de Linux es intencionalmente como la de UNIX. Así las lecciones aprendidas acerca de UNIX, incluyendo información sobre seguridad son aplicables tanto a UNIX como a Linux
- Linux es un sistema operativo estable y versátil, especialmente como un servidor de red.
- Proporciona un sólido entorno grafico, paquetes fáciles de instalar y aplicaciones de alto nivel.

4. Requerimientos de Hardware de Linux

Linux soporta plataformas de hardware tales como Intel x86, PowerPC, S/390, SPARC y Alpha. Los diferentes requerimientos de hardware para Linux se listan en la siguiente tabla

Hardware	Requerimientos
CPU	La serie x86 de Intel y sus compatibles, DEC, Alpha, Motorola, PowerPC, etc.
Tarjeta Madre (MotherBoard)	Sistemas de bus PCI, EISA, VESA y MCA.
Memoria	64 MB(mínimo), 256 MB recomendados para mayor eficiencia y ejecución
Monitor y Adaptador de Video	CGA, EGA, VGA, IBM monochrome, Súper VGA y otras tarjetas aceleradoras de video
Dispositivos de Puntero	Ratón serial estándar como Logitech, serie MM, Microsoft 2 botones, Sistemas Mouse de 3 botones, etc.
Controlador de Disco Duro	IDE, EIDE, MFM \, RLL y la mayoría de los controladores ESDI
Espacio de Disco Duro	Requiere un mínimo de 100 MB de espacio para una instalación mínima de Linux. Para una instalación completa con todos los servicios, los requerimientos pueden ser tanto como 2 GB
Unidades de CD-ROM	Sistema estándar de archivo ISO 9660 para CD-ROMS
Unidades de Cinta	SCSI
Impresoras	Impresoras paralelas
Modems	Modems seriales internos y externos
Tarjetas Ethernet	Soporta tarjetas Ethernet y adaptadores LAN populares

Tabla: Requerimientos de Hardware de Linux

5. Distribuciones de Linux

Como el código fuente para Linux fue desarrollado esta siendo distribuido gratuitamente, diferentes compañías han desarrollado sus propias versiones o *distribuciones* de Linux
Cada una de estas variedades tiene su propio conjunto de características, tales como procedimientos de instalación y administración, paquetes de software y configuraciones. Muchas de ellas están configuradas para un tipo específico de computadora.

Las 10 distribuciones principales se listan a continuación:

- Mandrake Linux, desarrollado por MandrakeSoft.
- Red Hat Linux, desarrollado por Red Hat
- Debian GNU/Linux, desarrollado por Debian.
- SuSE Linux, desarrollado por SuSe, Inc.
- Gentoo Linux, desarrollado por Gentoo Technologies, Inc.
- El Proyecto Slackware Linux, desarrollado por Slackware Linux, Inc.
- Lycoris Desktop, desarrollado por Lycoris
- Beehive Linux, desarrollado por el Equipo Beehive
- Caldera OpenLinux, desarrollada por Caldera Internacional, Inc.
- Turbolinux, desarrollado por Turbolinux, Inc.

Existen muchas más, aunque la distribución mas usada es Red Hat Linux

6. Organización de Linux

El sistema operativo Linux esta organizado funcionalmente en los siguientes tres niveles:

- Kernel (Núcleo)
- Shell (Interprete de Comandos)
- Herramientas y aplicaciones.

La representación esquemática de las tres partes principales del sistema operativo Linux se presenta en la Figura 1.3

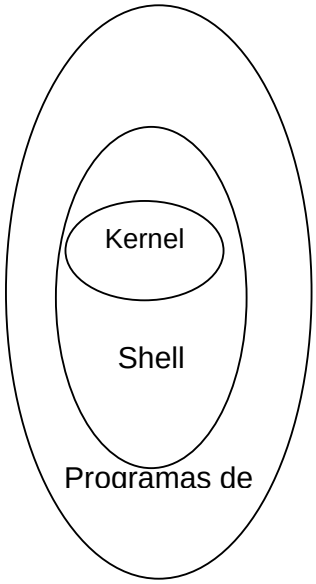

Figura 1.3: Partes Principales del Sistema Operativo Linux

El kernel

El kernel es el núcleo de un sistema operativo, así como la CPU es el núcleo de un sistema de computadora. El kernel es una colección de programas, la mayoría escritos en C y solo existe un kernel para cualquier sistema operativo. Este se comunica directamente con el hardware y sin un kernel un sistema operativo no puede interactuar con el hardware.

Algunas de las tareas importantes del kernel son:

- Verificar si el usuario es un usuario autorizado
- Hacer seguimiento de los diferentes programas que están ejecutándose y asignar un tiempo especifico a cada programa
- Asignar espacio de almacenamiento para los archivos en el sistema
- Ejecutar el programa shell

- Manejar la transferencia de información entre la computadora y las terminales

En un sistema multiusuario, a cada terminal se le asigna un numero y los usuarios trabajan en terminales conectadas a la computadora principal. El sistema operativo se comunica con la Terminal a través de los números de Terminal

El Shell

El sistema operativo Linux usa un shell para transferir los comandos desde el teclado a la computadora. El shell (interprete de comandos) es solo otro programa escrito en C.

Actúa como un intérprete entre los programas de los usuarios y el kernel. Traduce los comandos del usuario en la acción apropiada. El shell interactúa con el usuario, mientras que el kernel interactúa con el hardware de la maquina.

El shell es el programa que toma comandos y, ejecuta el programa apropiado o lo traduce en instrucciones que el kernel entiende. Por ejemplo, un comando tal como chdir será traducido por el shell a un formato entendible por el kernel, mientras que un comando tal como ls será ejecutado por el shell como un programa en el directorio /usr/bin/ls.

El shell es una interfaz basada en texto para el sistema Linux. En Linux, las interfaces graficas tales como el Sistema X Window (similar al presentado por Windows NT y 2000, que permiten al usuario ejecutar comandos usando el ratón y el teclado) también pueden ser usadas.

El shell por defecto en Linux es bash (Bourne Again Shell). Otros tipos de shell disponibles en Linux son:

- csh (C Shell)
- ksh (Korn Shell)
- sh (Shell)
- esch (enhanced C Shell)

Bourne Shell y el C Shell son los que se usan comúnmente. Ambos son controlados por comandos. El Korn Shell es el menos usado. Todos los shells sirven para el mismo propósito, pero tiene diferentes características y sintaxis

Herramientas y Aplicaciones

En el sistema Linux existe un cierto número de herramientas disponibles. Las herramientas son programas de usuario que pueden ser escritos por terceros para determinados tipos de aplicaciones. Típicamente, las herramientas se agrupan para realizar ciertas funciones, tales como programación, aplicaciones de negocio y procesamiento de texto.

A continuación se aprenderá acerca de la organización del sistema de archivos en el sistema operativo Linux

7. Sistema de Archivos en Linux

Todos los programas de usuario, documentos, herramientas, aplicaciones, etc son almacenados como archivos en sistema de computadora. Todos los archivos son almacenados en un dispositivo de almacenamiento secundario (usualmente un disco). Una porción del disco es separada para almacenar la información relacionada a los archivos almacenados. Esta unidad funcional se denomina *sistema de archivos*. Vea la figura 1.4

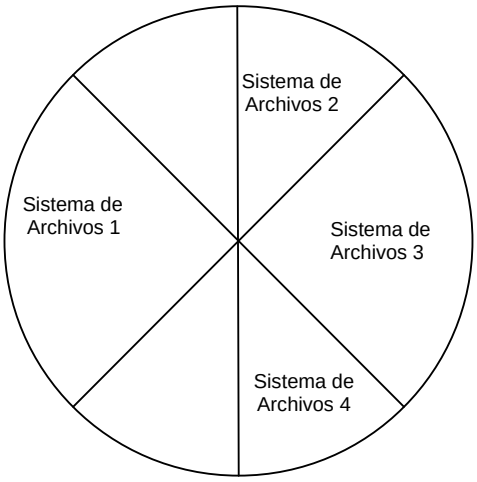

Figura 1.4 El Sistema de Archivos

Algunos de los puntos importantes aquí son:

- El área de la superficie donde se almacenan los archivos se dividen en pistas circulares.
- Las pistas circulares están divididas en sectores o bloques de disco (en forma de torta)
- Todos los bloques del disco son del mismo tamaño y tienen un numero único llamado el numero de bloque de disco
- El tamaño del bloque de disco varía dependiendo de la distribución Linux.

La mayoría de los sistemas modernos tienen un tamaño de bloque de disco de un 1 KB (kilobytes). Los bloques de disco están organizados en los siguientes cuatro grupos:

- Bloque de Arranque (Boot)
- Superbloque
- Bloque Inodo
- Bloque de Datos

Cada sistema de archivos tiene un bloque de arranque, un superbloque, una lista de bloques inodos y una lista de bloques de datos. La figura 1.5 ilustra la vista a nivel de bloques de disco de unidad de sistema de archivos. Los sistemas Linux típicamente usan más de una unidad de sistema de archivos.

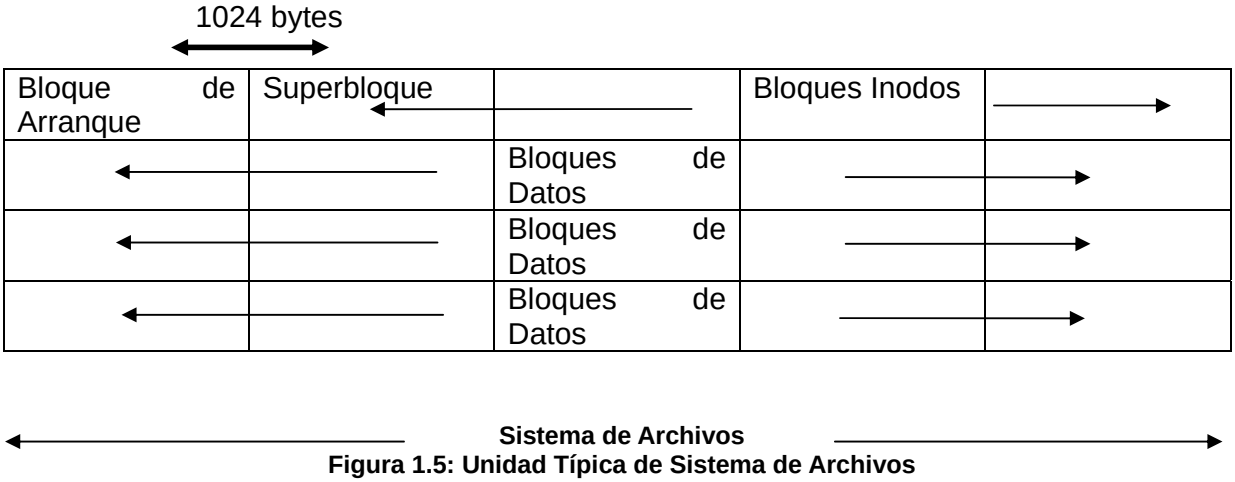

Figura 1.5: Unidad Típica de Sistema de Archivos

Ahora se va a entender la necesidad y uso de cada bloque en el sistema de archivos.

Bloque de Arranque

- Consiste de un bloque de disco que contiene el código para iniciar la computadora
- Ocupa el primer bloque de un sistema de archivos

- Un sistema solo requiere de un bloque de arranque para iniciar el sistema. En el resto de los sistemas de archivos, este bloque permanece vacío.

Superbloque

- Esta a continuación del bloque de arranque en el sistema de archivos
- Consiste de un bloque de disco que contiene información acerca del sistema de archivos
- Contiene información acerca del número de bloques en el sistema de archivos, el número de bloques asignados para inodos (se aprenderá acerca de los inodos más adelante) y el número de bloques que están actualmente libres.

Bloque Inodo

- Es el tercer grupo de bloques en un sistema de archivos.
- Contiene más de un bloque de disco para mantener información acerca de los archivos en el sistema de archivos.

Bloque de Datos

- Almacena el contenido del archivo
- Sigue a los bloques asignados para inodos.
- Un sistema de archivos contiene cierto número de bloques de datos.

Antes de proceder a aprender acerca de la partición del disco, se entenderá un poco mas acerca de la información que mantiene inodos.

Los inodos mantienen información acerca del propietario del archivo, los bloques de disco usados en el archivo, etc. Los archivos, por otro lado, contienen los datos del archivo. Asuma que se crea un archivo llamado midocumento.txt. Este debe ser almacenado en el disco. Dependiendo del sistema de archivos al que este asociado, los datos del archivo se almacenaran en los bloques de datos y otra información administrativa acerca de este, será almacenada en los bloques inodos.

Partición de Disco

Aquí, el sistema divide el disco en *particiones de disco*.

- Cada partición consiste en bloques, situados en forma contigua, pero separados de las otras particiones
- La partición puede ser un sistema de archivos o un *espacio de intercambio* (space swap).
- Un espacio de intercambio (space swap) se usa para implementar la memoria virtual, donde una porción de la memoria principal se almacena temporalmente
- La partición primaria es donde se almacenan los archivos relacionados al arranque.
- Las particiones del espacio de intercambio (space swap) son una secuencia lineal de bloques.
- El tamaño de los archivos cambia a través del tiempo (crece o disminuye)
- Un bloque de datos de un archivo puede no estar en una secuencia lineal de bloques. En vez de ello, puede estar disperso a lo largo de toda la partición.

8. Procesamiento de Texto

El sistema Linux proporciona métodos poderosos de procesamiento de un texto. Un ejemplo simple de procesamiento es encontrar el número de ocurrencias de un patrón dado en un texto. A continuación se considerara un ejemplo.

“La pronunciación de la palabra pronunciación es pronunciación”

En esta oración el patrón ‘pronunciación’ ocurre tres veces en el texto.

Cierto número de herramientas, tales como grep, egrep y fgrep, están disponibles para realizar el procesamiento de texto.

También existen otras herramientas de procesamiento de texto, que son conocidas como editores. Estos proporcionan las funcionalidades para crear, editar (modificar) y guardar texto. Algunos ejemplos de editores son:

- Vi: es conocido como editor visual y es el editor más popular. Vi es un programa que permite a los usuarios editar tanto archivos de texto como binarios. Los archivos de texto son aquellos que tienen caracteres alfanuméricos, mientras que los binarios contienen caracteres entendibles por la maquina. A pesar que también puede leer archivos binarios, vi es conocido normalmente como un editor de texto.
- Un editor de texto es como un procesador de palabras. Los editores de texto se usan principalmente para escribir programas, que luego son convertidos en un código entendible por la maquina a través de otro programa.
- Ed: mientras que vi es un editor que permite visualizar el contenido de un archivo, una pantalla a la vez, ed es un editor en linea. En cualquier momento, solo puede mostrarse una linea del archivo.
- Sed: es un editor de flujos basado en ed. Puede editar archivos sin intervención del usuario. Los comandos de edición pueden ser pasados como argumentos de linea de comandos. Los argumentos de linea de comandos son aquellos que se proporcionan junto con un comando, antes de que el comando sea ejecutado. Estos argumentos van como entradas sobre las cuales el programa puede trabajar para producir cierta salida.
- Sed se utiliza extensivamente en los sistemas Linux. Proporciona mecanismos poderosos para editar flujos de datos pasados a el como entrada
- Emacs: es un poderoso editor de texto. Siendo C el lenguaje de programación usado comúnmente en los sistemas Linux, emacs tiene características incorporadas que permite dar formato automático a los programas C, a su vez también permite la búsqueda de patrones y lectura de correo electrónico desde el editor.

A continuación se presenta una breve discusión sobre las capacidades de programación y documentación disponibles en Linux.

9. Programación

Se puede programar a través del shell y esto se conoce como *programación de shell*
Linux proporciona más de un shell. El Bourne Again Shell (popularmente conocido como bash) es el shell mas usado. Se aprenderá acerca de los diferentes shells que proporciona Linux en la unidad 2: El Sistema Linux.
Cada shell en Linux proporciona la capacidad de programación. Un programa shell puede invocar las herramientas proporcionadas en Linux a través de una sintaxis simple. La programación de shell es similar a un lenguaje de programación como C. pero con una sintaxis diferente.
Combinado con el poder del procesamiento de texto, la programación en Linux es extremadamente poderosa. La administración de sistemas complejos en sistemas operativos estilo UNIX se hace típicamente a través de la programación del shell. El administrador del sistema usa la programación del shell en forma extensiva para administrar y monitorear el sistema operativo.

10. Documentación.

Linux proporciona una documentación bastante elaborada para todas sus herramientas. Las herramientas son referidas comúnmente como comandos.
El sistema Linux proporciona cierto número de comandos. Algunos de ellos son:

- clear – limpia la pantalla
- date – muestra la fecha y hora
- cal – muestra el calendario del mes actual
- who – muestra los usuarios que están actualmente conectados al sistema

Sin embargo, dado que los comandos son tan extensos, no es posible para ningún usuario recordar todos los comandos y la sintaxis asociada con estos.
La documentación puede ser leída fácilmente con la ayuda de una herramienta controlada

por comandos llamada man.
A continuación se dan ejemplos de su uso. El comando man será discutido en detalle en la unidad 2: El Sistema Linux.

man clear
man date
man man

Los primeros dos usos del comando man muestran la documentación para los comandos clear y date, respectivamente. El tercer uso, muestra la documentación del mismo comando man

11. Características del Sistema Linux

El sistema Linux ofrece las siguientes características:

- Estabilidad: tiene protección de la memoria entre procesos, de manera que uno de ellos no pueda colgar el sistema.
- Multitarea: varios programas (realmente procesos) ejecutándose al mismo tiempo
- Multiplataforma: se ejecuta en muchos tipos de CPU, no solo Intel
- Multiusuario: varios usuarios en la misma maquina al mismo tiempo (y sin licencias para todos)
- Manejo de la Memoria: la memoria gestiona como un recurso unificado para los programas de usuario y para cache de disco, de tal forma que toda la memoria libre puede ser usada para cache y este puede a su vez ser reducido cuando se ejecutan grandes programas
- Interfaz Grafica de Usuario: KDE, GNOME.
- Desarrollo de Software: KDevelop (Lenguaje C, C++, Java, PHP, Perl, Phyton, entre otros)
- Trabajo de redes: TCP/IP, incluyendo ftp, telnet, NFS, Gíreles, etc.
- Disponibilidad del código Fuente: todo el código fuente esta disponible, incluyendo el núcleo completo, todos los drivers, las herramientas de desarrollo y todos los programas de usuario; además están disponibles libremente.

12. Software disponible en Linux

En Linux se encuentran disponibles varios tipos de software. Algunas de las categorías son:

- Aplicaciones
- Software de Desarrollo
- Software Científico
- Software de Sistema
- Utilitarios
- Juegos.

Unidad 2: El Sistema Linux

Objetivos de Aprendizaje

Al finalizar esta unidad, Ud será capaz de:

- Explicar el procedimiento de ingreso/salida en un sistema operativo Linux
- Discutir el uso del formato de comandos Linux
- Describir la redirección de entrada y salida
- Explicar el uso de algunos comandos básicos de Linux
- Discutir el uso de tuberías (pipes) y filtros.

1. Introducción

En la unidad 1: Fundamentos de Linux, se hizo una breve introducción al sistema operativo Linux. Linux es un sistema operativo multiusuario y multitarea, donde múltiples usuarios pueden ejecutar múltiples aplicaciones en forma simultanea en una única computadora centralizada (solo un procesador)
Linux cumple con POSIX (Portable Operating System Interface for UNIX) y es un sistema operativo estable y versátil. A traves del curso, se usaran UNIX y Linux indistintamente.
A continuación se aprenderán a usar las poderosas características del sistema

2. Ingreso y Salida

Los usuarios en un sistema operativo multiusuario trabajan en terminales, que comprenden una unidad de salida (también conocida como monitor) y un teclado. Los terminales tienen un número único asociado con ellos y están conectados a la unidad principal, que no necesariamente reside en la misma ubicación.
En un sistema que permite que múltiples usuarios operen en forma concurrente, debe existir una única forma de identificar a los usuarios. El sistema también debe asegurar que un usuario no suplante a otro y obtenga acceso a archivos confidenciales. Todo esto se logra a traves del procedimiento de *ingreso (login)*
Algunos puntos a considerar acerca del procedimiento de *ingreso* son:

- Cada usuario del sistema es identificado con una cuenta única
- Los usuarios son identificados por su nombre de registro (login name) y sus cuentas están protegidas a traves de contraseñas de usuario (user password)
- Cada sistema Linux tiene un administrador de sistema que administra, monitorea y maneja el sistema.
- Uno de los trabajos del administrador de sistema es el crear cuentas proporcionar *nombres de registro* para los usuarios
- también proporciona la contraseña inicial a cada cuenta
- Root es un usuario de sistema en Linux que tiene el control completo del sistema operativo. Cuando se ingresa como *root*, el administrador del sistema puede realizar cualquier tarea en el sistema operativo. El root también es conocido como superusuario.
- Cada sistema tiene un *nombre de servidor (host name)* asignado. El nombre de servidor facilita la rápida identificación de una maquina en una red

Se usara la consola (consolé) BASH para KDE y SuSE Linux 10.3
Ahora se aprenderá a ingresar al Sistema Operativo Linux

Ingreso

Una solicitud típica de ingreso y registro en un sistema Linux es como sigue:

Login:

El usuario tiene que ingresar el nombre de registro (login name). Posteriormente solicita el ingreso de la contraseña (password)

Login:
Password:

El usuario debe ingresar con mucho cuidado la contraseña (password), dado que los caracteres ingresados no son mostrados en pantalla. Si la contraseña se ingresa de manera incorrecta, aparecerá el siguiente mensaje de error:

Login Incorrecto

Todos los sistemas operativos similares a UNIX, incluyendo Linux, son sensibles a mayúsculas y minúsculas.
Una vez que tanto el nombre de ingreso como la contraseña se ingresan correctamente, el usuario *ingresa* en el sistema. En este ejemplo, cuando se ingresa exitosamente, se estará en un directorio llamado:

/home/nombre_de_usuario (en mi caso edwar, /home/edwar)

Esto se llama *directorio home del usuario*. La primera barra diagonal "/" indica el directorio raíz y las subsecuentes solo son separadores para directorios. Los archivos y directorios se estudian en la Unidad 4 – *La Estructura de Archivos de Linux.* Aquí será suficiente con establecer que los directorios son lugares de almacenamiento para archivos.
El sistema presentara una pantalla y una solicitud de comandos, donde se puede ingresar comandos. La solicitud de comandos visible en la pantalla es:

usuario@nombre_del_servidor (en mi caso edwar@localhost:~>).

El nombre de ingreso edwar, que aparece por segunda vez, indica que se esta en el directorio home de edwar. Se aprenderá más respecto a esto en la Unidad 4 – *La Estructura de Archivos de Linux*
El signo > al final de la cadena de solicitud, indica que se esta en el BASH y esta esperando alguna entrada del usuario. Para diferenciar un usuario normal de superusuario, el sistema operativo Linux coloca el símbolo # en lugar de > para el superusuario.

Salida

Antes de terminar la sesión con el sistema, es importante que se *salga* del sistema. Esto previene que otros usuarios puedan hacer un mal uso de los archivos.
Esto se logra ingresando cualquiera de los siguientes comandos:

- logout
- exit
- <Control + D> (mantener presionado la tecla Control y presionar la tecla D)

3. Formato de Comandos

Al momento de escribir un programa en C, se usara un editor de texto. Para compilar un programa en C a su forma ejecutable, se usa un compilador de C. Estos son programas que realizan tareas específicas. A nivel del sistema operativo, estos programas se llaman *comandos.*
Un comando es un programa que realiza una tarea específica. Todas las tareas realizadas en un sistema Linux pueden llevarse a cabo a traves de comandos. Una linea de comandos puede ser estructurada en tres partes:

- **El nombre del comando:** es el nombre del comando que realiza una tarea especifica
- **Opciones:** a la mayoría de los comandos se les puede dar una o mas opciones para hacer que los resultados sean mas específicos
- **Argumentos:** son usualmente archivos o directorios de donde los datos pueden ser leídos para que el comando opere sobre ellos.

La sintaxis de una linea de comandos se da a continuación:

comando –opcion$_1$ –opcion$_2$ –opción $_{...}$ arg$_1$ arg$_{2\ ...}$ arg$_n$

Aquí *comando* es el nombre del comando; de opcion$_1$ a opción es el rango de las n opciones posibles para dicho comando y de arg$_1$ a arg$_n$ es el rango de m argumentos que pueden ser pasados al comando
La forma mas simple de un comando es aquella que solo tiene un nombre de comando, por ejemplo, un comando *passwd* o *clear*. Cuando se ingresa por primera vez en un sistema
Linux, normalmente se usa la contraseña establecida por el administrador del sistema. Una de las primeras cosas que se debe hacer al ingresar con éxito por primera vez, es cambiar la contraseña inicial.
Dado que existen cientos de comandos y la mayoría de ellos tiene varias opciones disponibles, no es posible recordar todos los comandos y sus opciones. Linux proporciona un manual que puede ejecutarse en el prompt de comandos. El comando es llamado *man*, que viene de **man**ual pages (paginas del manual)
En las siguientes secciones y unidades posteriores, entre las etiquetas **Uso de Comando y Fin de Uso de Comando**, se mostrara como se usa un comando en Linux, con una

breve explicación del mismo. En algunos casos, también se mostrara el uso de algunas opciones del comando.
Se asume que el directorio home es /home/nombre_de_usuario (en mi caso edwar, /home/edwar). Se ingresaran todos los comandos como se haría en un entorno Linux. Por ejemplo, para mostrar el uso del comando *man*, se hará lo siguiente

edwar@localhost:~> man date

Siguiendo esto, se mostrara la salida o resultado del comando finalmente se terminara con la cadena del prompt, que es nuevamente edwar@localhost:~>

Uso de Comando 2.1

Se asumirá que la contraseña inicial establecida por el administrador es mm123. El comando *passwd* se usa para cambiar la contraseña del usuario.

edwar@localhost:~> **passwd**

Cambiando contraseña para edwar (en ingles seria: changing password for edwar)
Contraseña anterior: mm123 (provista por el administrador del sistema)
Nueva contraseña: ~~eduardo1977~~
Vuelva a introducir la nueva contraseña: ~~eduardo1977~~
Contraseña cambiada para: edwar

Cuando se ingresa el comando *passwd*, aparece el mensaje “Changing password for edwar” (Cambiando la contraseña para edwar), seguido del prompt para ingresar la nueva actual. Cuando se ingresa la contraseña actual (mostrada con letras tachadas para indicar que estos caracteres no aparecen en pantalla) y se determina que es correcta, el sistema solicita al usuario que ingrese la contraseña nueva.
Normalmente, todos los sistemas siguen ciertas convenciones para seleccionar contraseñas. Si la nueva contraseña ingresada no esta conforme a estas convenciones, aparecerá uno de los siguientes mensajes

Contraseña Incorrecta: demasiado parecido a la anterior
Contraseña Incorrecta: demasiado corta
Contraseña Incorrecta: demasiado sencilla.

El sistema solicita al usuario que reingrese la nueva contraseña, si la nueva contraseña es correcta. Si las entradas coinciden, se establece la nueva contraseña. Caso contrario, se muestra el mensaje “Las contraseñas no coinciden” y vuelve a pedir una nueva contraseña hasta un numero de intentos considerables. Cada que se ejecuta por completo, con o sin mensajes de error, el usuario retorna al prompt de comandos (command prompt)

Fin de Uso de Comando 2.1

Ahora que se ha entendido como se usa un comando simple en el sistema Linux, se vera el uso de comando con un argumento y sin opciones.

Uso de Comando 2.2

El comando *man* muestra la documentación de un comando que se pasa como argumento

edwar@localhost:~> **man** clear

clear(1) clear(1)

NAME
clear – clear the terminal screen

SYNOPSIS
clear

DESCRIPTION
clear clears your screen if this possible. It looks in the environment for

the terminal type and then in the terminfo database to figure out how to clear the screen.

SEE ALSO

tput(1), terminfo(5) clear(1)

edwar@localhost:~>

La forma mas simple de un comando man solo tiene un argumento. El argumento es el nombre del comando del cual se quieren las páginas del manual (o documento). En el ejemplo dado se obtienen las paginas de manual para el comando clear.

Se puede salir de las paginas man ingresando la letra q, que viene de quit (salir)

Si se ha ingresado edwar@localhost:~> man sin un argumento, aparecerá el siguiente mensaje: ¿Cuál pagina de manual desea?

edwar@localhost:~>

Fin de Uso de Comando 2.2

Un comando puede usarse con o sin opciones. Sin embargo, muchos comandos en Linux tienen opciones. Un ejemplo de estos comandos es el siguiente:

Uso de Comando 2.3

El comando **cat** se usa para concatenar o unir archivos.

Sintaxis de cat: cat [opciones] [archivos]

edwar@localhost:~> cat miarchivo.txt

Este es un archivo de ejemplo.
Creado el 21 de marzo de 2009
Hora de la creación 17:15

edwar@localhost:~>

En el ejemplo anterior el archivo miarchivo.txt debe estar creado, en caso contrario aparecerá un mensaje que dice “cat: miarchivo.txt: No existe el fichero o el directorio.
Cat es un comando que acepta uno o mas nombres de archivos, con**cat**enándolos (uniéndolos) y mostrándolos en pantalla. Asuma que se ha ingresado lo siguiente:

edwar@localhost:~> cat miarchivo.txt miarchivo.txt

La salida será como sigue

Este es un archivo de ejemplo.
Creado el 21 de marzo de 2009
Hora de la creación 17:15
Este es un archivo de ejemplo.
Creado el 21 de marzo de 2009
Hora de la creación 17:15

edwar@localhost:~>

El comando simplemente une los dos archivos y los muestra. También cat puede usarse sin argumentos. Asuma que se ha ingresado lo siguiente:

edwar@localhost:~> cat

El programa espera alguna entrada, para que pueda concatenar los datos de entrada y mostrarlos. Dado que no se proporciono argumento, cualquier cosa que se ingrese en el teclado será mostrado. La entrada se toma del teclado, que es el dispositivo de entrada estándar.
Se vera otro ejemplo.

edwar@localhost:~> cat –n miarchivo.txt

Este es un archivo de ejemplo.
Creado el 21 de marzo de 2009
Hora de la creación 17:15

edwar@localhost:~>

En el ejemplo anterior, se uso la opción –n. Las opciones para un comando son precedidas por un signo – (menos). Este signo indica al comando que lo que sigue no es un argumento sino una opción. Todos los comandos en Linux son de una palabra de longitud y están en letras minúsculas.
La opción –n muestra los números de linea contra cada linea. –n viene de **n**umero. También es posible combinar opciones con un solo signo – precediendo a todas las opciones como se muestra a continuación:

edwar@localhost:~> cat –ns miarchivo.txt

Fin de Uso de Comando 2.3

Los comandos en Linux pueden tomar opciones y/o argumentos. Las opciones son aquellas que realizan una tarea especializada y siempre están precedidas por un signo – (menos). En el ultimo ejemplo del Uso de Comando 3 –ns son opciones para el comando cat. Algunos comandos en Linux requieren un argumento, normalmente el nombre de un archivo. Si no se proporciona, entonces se asumirá la entrada del usuario. Las opciones y argumentos pueden ser opcionales. Para determinar cuales son opciones y cuales obligatorias, se puede hacer uso de man. Cuando se ve la ayuda de un comando usando el comando man, si las opciones o argumentos están encerrados en [ ] (corchetes), entonces son opcionales. Caso contrario, son obligatorios.

4. Entrada Estándar, Salida Estándar y Error Estándar.

En el uso de comando anterior, se hace mención al termino 'dispositivo de entrada estándar'. Ahora se discutirá brevemente como trabajan los sistemas Linux con la entrada, salida y errores. En todos los sistemas Linux, cualquier programa (incluyendo todos los comandos Linux), esta conectado automáticamente a tres archivos. Estos son:

- Entrada estándar
- Salida estándar
- Error estándar.

Obsérvese la figura 2.1

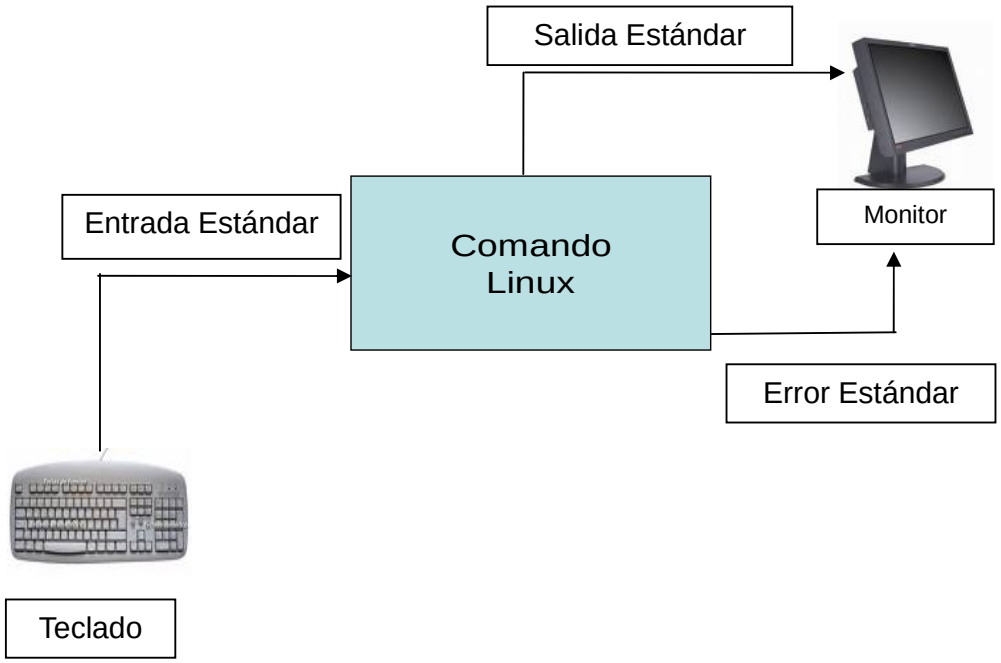

Figura 2.1 Linux y la Entrada, Salida y Error Estándar

Un programa obtiene su entrada estándar, envía su salida a la salida estándar y los mensajes de error al error estándar. Todos ellos por defecto en el terminal. En un sistema Linux, todos los dispositivos son tratados como archivos.
Considere el siguiente comando:
edwar@localhost:~> cat.
Este espera una entrada desde el teclado, porque el teclado es la entrada estándar.
Considere otro comando:
edwar@localhost:~> cat miarchivo.txt.
La entrada estándar ahora es el archive miarchivo.txt y no el teclado. Pero la salida estándar sigue siendo la pantalla

5. Redirección de Entrada/Salida

Se puede hacer un programa que tome la entrada de un archivo y envíe la salida hacia un archivo a traves de la *redirección*. La redirección es una manera a traves de la cual se puede cambiar, tanto la entrada como la salida estándar. Algunos puntos acerca de la redirección se mencionan a continuación:

- Se pueden usar los operadores de redirección > y < para salida y entrada respectivamente
- El operador > hace del archivo que sigue al operador la nueva salida estándar y de manera similar, el operador < cambia la entrada estándar.
- El uso de los operadores > y < es temporal
- Cuando el comando finaliza, se establecen nuevamente los valores por defecto

Uso de Comando 2.4
El operador **>** se usa redirigir la salida a la nueva salida estándar. La nueva salida estándar es el archivo que sigue al símbolo >
edwar@localhost:~> cat > nuevoarchivo.txt
Este es el primer archivo que he creado.
Lo he creado el 21 de marzo de 2009
Ctrl. –d
edwar@localhost:~>

Dado que no se ha ingresado ningún archivo después del comando cat, la entrada por defecto es el teclado. Cualquier cosa que ingrese a traves del teclado, es redirigida hacia el archivo de salida nuevoarchivo.txt. Se finaliza la entrada presionando ctrl. –d. <ctrl. –d> no será visualizado en la pantalla como se muestra, dado que es una tecla que presiona el usuario para finalizar la entrada. Aquí se muestra para indicar que es la forma de terminar la entrada.
Vea las siguientes líneas

edwar@localhost:~> cat
Este es el primer archivo que he creado.
Este es el primer archivo que he creado.
Lo he creado el 21 de marzo de 2009
Lo he creado el 21 de marzo de 2009
Ctrl. –d
edwar@localhost:~>

Las líneas que se ingresan se muestran inmediatamente en la pantalla como un eco, ya que tanto la entrada como la salida estándar usan el terminal (monitor)

edwar@localhost:~> cat < miarchivo.txt >nuevoarchivo.txt

El uso anterior toma la entrada de miarchivo.txt y envía la salida hacia nuevoarchivo.txt.
El nuevoarchivo.txt se sobrescribe ahora con las líneas de miarchivo.txt. El operador > establece al archivo que le sigue de 0 bytes, si este ya existe. El archivo se llena nuevamente con los nuevos datos.
Algunas veces se puede estar interesado en añadir datos a un archivo ya existente. Esto posible usando el operador >>, el cual añade los datos al archivo si ya existe, o sino, crea

un nuevo archivo y escribe los datos.
Por añadir, se refiere a que los datos son siempre agregados al archivo ya existente.
Dado que nuevoarchivo.txt ha sido sobrescrito, se recrea con las dos líneas originales.

```
edwar@localhost:~> cat
        Este es el primer archivo que he creado.
        Lo he creado el 21 de marzo de 2009
        Ctrl. –d
edwar@localhost:~>
```

Ahora se vera como se agrega usando el comando cat.

```
edwar@localhost:~> miarchivo.txt >> nuevoarchivo.txt
edwar@localhost:~> cat –n nuevoarchivo.txt
        Este es el primer archivo que he creado.
        Lo he creado el 21 de marzo de 2009
        Este es un archivo de ejemplo.
        Creado el 21 de marzo de 2009
        Hora de la creación 17:15
edwar@localhost:~>
```

Las líneas de la 3 a la 5 han sido añadidas a nuevoarchivo.txt

```
edwar@localhost:~> cat archivo1 archivo2 >> archivo3.txt
```

Los contenidos de archivo 1 y 2 son concatenados. Los datos concatenados son luego añadidos a archivo3.txt. Los archivos, archivo 1 y 2 no se modifican
Se debe ser cuidadoso al usar la redirección. Algunas veces, se puede usar el archivo de salida como un archivo de entrada durante la redirección.
Por ejemplo considérese el siguiente comando:

```
edwar@localhost:~> cat archivo1 archivo2 >> archivo2.txt
```

El sistema reportara un error.

```
cat: archivo2: Los ficheros de entrada y salida son el mismo.
```

Mientras se muestre este mensaje de error para indicar que se esta usando el mismo archivo como entrada y salida, el contenido de archivo1 es añadido a archivo2. La salida del uso anterior es la misma que la obtenida usando el comando dado a continuación:

```
edwar@localhost:~> cat archivo1 >> archivo2.txt
```

Fin de Uso de Comando 2.4

6. Comandos Simples

Una vez entendida la sintaxis de comandos y la redirección de entrada y salida se aprenderán algunos comandos simples. Muchos comandos en Linux trabajan sobre archivos y directorios. Estos serán discutidos con cierto detalle en la Unidad 4: La Estructura de Archivos de Linux de este volumen
Se dará la sintaxis básica de los siguientes comandos Linux:

- cal
- date
- head
- tail
- sort
- cmp
- wc
- grep
- pr
- cut

Uso de Comando 2.5

El comando **cal** se utiliza para mostrar el calendario. El comando mostrara el calendario para el mes, año, etc, basado en las opciones y argumentos proporcionados

Sintaxis de cal: cal [opciones] [mes] [año]
edwar@localhost:~> cal

marzo 2009

do	lu	ma	mi	ju	vi	sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

edwar@localhost:~>

Como se entiende hasta ahora, cal es el comando dado para mostrar el calendario. El comando cal sin ningún argumento muestra el calendario del mes actual

edwar@localhost:~> cal –y

Esto muestra el calendario completo para el año actual

edwar@localhost:~> cal –m

Esto muestra el calendario para el mes actual con el Lunes como primer día de inicio de la semana, el Domingo es el día de inicio por defecto

edwar@localhost:~> cal 12 2009

Esto muestra el calendario del mes de Diciembre del año 2009

edwar@localhost:~> cal 12

Esto muestra el calendario para el año 12
Si se tienen dos argumentos, cal considera el primero como el mes y el siguiente como el año. Si se tiene solo un argumento, este se considera automáticamente como el año

Fin de Uso de Comando 2.5

Uso de Comando 2.6

El comando **date** se usa para conocer la fecha y hora actual.
Sintaxis de date: date [opción] [+formato]
edwar@localhost:~> date
lunes 23 de marzo de 2009 15:33:05
edwar@localhost:~>
El comando date sin argumentos muestra el día, el mes, fecha, hora y año actual.
edwar@localhost:~> date +%H
15
edwar@localhost:~>
date con el formato %H muestra solo la hora

Vea las páginas de man para la lista de opciones y diferentes formatos en que se puede mostrar la fecha. El comando date también puede usarse para establecer la fecha y hora del sistema. Vea las páginas de man para información sobre como establecer la fecha y hora del sistema.

Fin de Uso de Comando 2.6

Para los siguientes comandos como: head, tail, sort, cmp y grep se usaran archivos (previamente creados), nombres.txt y lineas.txt. El archivo nombres.txt contiene los siguientes nombres:

noddy
tom
jerry
donald
mickey
asterix
tintin
obelix

El archivo lineas.txt contiene las 11 líneas siguientes, que son citas bien conocidas de personas famosas:

1. Una mentira nunca vive para ser vieja. –Sófocles
2. Un hombre honesto es siempre un niño. –Martial
3. Cuando estés en duda, di la verdad. –Mark Twain
4. Las palabras falsas no son malas en si mismas, sino que infectan el alma con maldad. –Sócrates
5. Toda la verdad no es para ser dicha todas las veces. –Thomas Fuller M.D.
6. No me importa la mentira, pero odio la inexactitud. –Samuel Butler
7. Me hago a mi mismo un daño mayor mintiendo, que aquel que ocasiono a quien le ha dicho una mentira. –Michel de Motaigne
8. Grandes esperanzas hacen grandes hombres. – Thomas Fuller
9. Una mente fuerte siempre tiene esperanzas, y tiene siempre razón para estar esperanzada. –Polybius
10. La Esperanza es como un sebo, cubre cualquier anzuelo. –Ben Jonson
11. Aquel que tiene salud, tiene esperanza; y aquel que tiene esperanzas lo tiene todo. –Proverbio Árabe.

Se usara el archivo anterior para entender los comandos head y tail. Estos son útiles cuando se quiere ver partes de un archivo.

Existen dos puntos importantes que se deben notar aquí:

- Los números de linea dados por lineas.txt no son parte del texto. Es simplemente para entender los comandos con claridad.
- Una linea es definida como una secuencia de caracteres delimitada por un carácter de cambio de linea. Así una linea puede estar en mas de una fila

Uso de Comando 2.7

El comando **head** se utiliza cuando se desea ver solo unas cuantas líneas de un archivo, comúnmente las primeras líneas. El comando muestra la primera parte de un archivo

Sintaxis de head: head [opcion] [archivo]

edwar@localhost:~> head lineas.txt

Esto mostrara las primeras 10 líneas del archive lineas.txt. el valor por defecto es 10

edwar@localhost:~> head -4 lineas.txt

1. Una mentira nunca vive para ser vieja. –Sófocles
2. Un hombre honesto es siempre un niño. –Martial
3. Cuando estés en duda, di la verdad. –Mark Twain
4. Las palabras falsas no son malas en si mismas, sino que infectan el alma con maldad. –Sócrates

edwar@localhost:~>

Esto muestra las primeras 4 líneas del archivo

Fin de Uso de Comando 2.7

Ahora se vera como utilizar el comando tail usando el mismo archivo.

Uso de Comando 2.8

El comando **tail** se utiliza para ver las últimas líneas de un archivo. Combinando head y tail inteligentemente, también se podrán ver las líneas en el medio de un archivo. Para hacer esto, se necesita entender acerca de tuberías y filtros, que serán discutidos mas adelante en esta unidad.

Sintaxis de tail: tail [opcion] [archivo]

edwar@localhost:~> tail lineas.txt

El comando dado imprime las 10 ultimas líneas

edwar@localhost:~> tail –c10 lineas.txt

bio Arabe

edwar@localhost:~>

Esto muestra los 10 últimos bytes (caracteres) del archivo

La opción –c seguida por el numero de bytes a mostrar, le indica al comando tail que muestre los últimos n bytes del archivo. Se puede estar preguntando por que solo 9 caracteres son visibles, cuando el número que se ingreso fue 10. El carácter de nueva linea esta presente al final, el cual no es visible como un carácter.
El hecho de que edwar@localhost:~> se muestra en una linea separada es el indicativo de un carácter de nueva linea presente al final
Compárese esto con head c-10

edwar@localhost:~> head –c10 lineas.txt

1. Una menedwar@localhost:~>

Debido a que el carácter 11 no es un cambio de linea, la cadena del "prompt" edwar@localhost:~> se muestra en la misma linea

Fin de Uso de Comando 2.8

A continuación se estudia como se puede realizar un ordenamiento usando un comando simple en Linux. El poder de los sistemas operativos UNIX, es que muchas de las funcionalidades se proporcionan como simples comandos.

Uso de Comando 2.9

El comando **sort** se usa para ordenar el contenido del archivo y mostrar la lista ordenada en la salida estándar.

Sintaxis de sort: sort [opciones] [archivo]

```
edwar@localhost:~> sort nombres.txt
        asterix
        donald
        jerry
        mickey
        noddy
        obelix
        tintin
        tom
edwar@localhost:~>
```

Esto ordena las líneas del archivo nombres.txt y muestra los datos ordenados.

edwar@localhost:~> sort –r nombres.txt

Esto ordena las líneas del archivo en orden inverso (descendiente)

Fin de Uso de Comando 2.9

Uso de Comando 2.10

El comando **cmp** se utiliza para comparar el contenido de dos archivos. El número del carácter y el número de linea donde ocurrió la primera diferencia, los cuales se muestran en la salida estándar. Este comando es útil cuando se quiere comparar y encontrar si los dos archivos son idénticos.

Sintaxis de cmp: cmp [opción] archivo1 archivo2

Las únicas opciones permitidas para cmp son –l y –s

```
edwar@localhost:~> cmp lineas.txt nombres.txt
nombres.txt lineas.txt son distintos: byte 1, linea 1
edwar@localhost:~>
```

Este comando compara dos archivos y muestra el carácter y el número de linea donde la primera diferencia ocurrió. Se nota que en nombres.txt y en lineas.txt el primer carácter es diferente en ambos archivos
La opción –l muestra el numero de byte (en decimal) y el valor de los bytes diferentes (en octal) para cada diferencia. La opción –s no muestra nada para los archivos con diferencias y retorna un estado de salida. Cuando se escriben programas, se pueden usar los comandos disponibles en el shell. Cuando los comandos retornan como estado de salida, es similar al valor retornado por una función. Esto puede ser usado de forma apropiada.

En el archivo1.txt se tiene:

abcd
efgh
ijkl
nopq
mnop

En el archivo2.txt se tiene:

abcde
egh
ijkl
nopq
rstu

Se va a intentar la opción –l y ver que sucede.

```
edwar@localhost:~>  cmp –l archivo1.txt archivo2.txt
     5      12     145
     6      145    12
     7      146    145
     16     155    156
     17     156    157
     18     157    160
     19     160    161
cmp: fin de fichero encontrado en archivo1.txt
edwar@localhost:~>
```

El 5º carácter en el primer y segundo archivo difieren. 12 y 145 son los números octales de los caracteres. 12 en octal es 10 en decimal y 145 en octal es 101 en decimal. 10 es el valor ASCII del carácter de cambio de linea y 101 es el valor ASCII de la letra “e”.
El sistema numérico octal se utiliza normalmente en computadoras para representar números. El fin de fichero encontrado en archivo1.txt dice que el fin del archivo ha sido alcanzado en archivo1.txt

Fin de Uso de Comando 2.10

Uso de Comando 2.11

Wc viene de **w**ord **c**ount (contador de palabras). El comando **wc** se usa encontrar el número de caracteres, el número de palabras y el número de líneas en un archivo

Sintaxis de wc: wc [opciones] [archivo]

```
edwar@localhost:~>  wc nombres.txt
     8      8      52 nombres.txt
edwar@localhost:~>
```

El comando anterior muestra el numero de líneas, palabras y caracteres (en ese orden) de nombres.txt

```
edwar@localhost:~>  wc nombres.txt lineas.txt
     8      8      52 nombres.txt
     11     142    707 lineas.txt
     19     150    759 total
edwar@localhost:~>
```

Esto muestra el numero de líneas, palabras, y caracteres de ambos archivos, uno a continuación del otro y también el total

```
edwar@localhost:~>  wc –l nombres.txt
     8 nombres.txt
edwar@localhost:~>
```

La sentencia anterior solo muestra el numero de líneas

```
edwar@localhost:~>  wc –w nombres.txt
     8 nombres.txt
```

edwar@localhost:~>

La sentencia anterior solo muestra el numero de palabras

edwar@localhost:~> wc –lc nombres.txt

8 52 nombres.txt

Esto muestra el numero de líneas y los caracteres

Fin de Uso de Comando 2.11

Una de las características interesantes de comandos Linux, es la familia de comandos de búsqueda de patrones. Se llamas una familia de comandos, porque tiene comandos que proporcionan la funcionalidad básica de búsqueda de patrones con algunas características adicionales. Se aprenderá como realizar una búsqueda de patrones utilizando el comando grep.

Uso de Comando 2.12

La búsqueda de patrones, es una facilidad importante de los sistemas Linux. El comando **grep** se utiliza para hallar un patrón dado en un archivo.

Sintaxis de grep: grep [patrón] [archivo]

Grep es uno de los más poderosos comandos en Linux. Este comando compara el patrón dado con cada linea del archivo y muestra todas las líneas donde se encuentre el patrón. Este es el uso mas simple de grep

edwar@localhost:~> grep mentira lineas.txt

1. Una mentira nunca vive para ser vieja. –Sófocles
6. No me importa la mentira, pero odio la inexactitud. –Samuel Butler
7. Me hago a mi mismo un daño mayor mintiendo, que aquel que ocasiono a quien le ha dicho una mentira. –Michel de Motaigne

edwar@localhost:~>

Note que solo se muestran 3 líneas, en las cuales ocurre el patrón “mentira”. El comando grep busca patrones. Considere el código a continuación:

edwar@localhost:~> grep esper lineas.txt

8. Grandes esperanzas hacen grandes hombres. – Thomas Fuller
9. Una mente fuerte siempre tiene esperanzas, y tiene siempre razón para estar esperanzada. –Polybius
11. Aquel que tiene salud, tiene esperanza; y aquel que tiene esperanzas lo tiene todo. –Proverbio Árabe

edwar@localhost:~>

Ahora se ve que las líneas en las cuales el patrón “esper” ocurre han sido mostradas. Note que la linea “La Esperanza es como un cebo, cubre cualquier anzuelo.—Ben Jonson” no es parte de la salida. ¿Puede adivinar por que?, grep realiza una comparación considerando mayúsculas y minúsculas y de acuerdo al patrón “esper” no es lo mismo que el patrón “Esper”. Se puede instruir al grep para que se ignore esto usando la opción –i. El comando grep tiene muchas opciones. Refiérase a las páginas de man para más detalle acerca de ellos.

Fin de Uso de Comando 2.12

Uso de Comando 2.13

El comando **pr** se usa para convertir un archivo de texto para su impresión.

Sintaxis pr: pr [opción] [archivo]

Pr convierte archivos de texto para su impresión. La opción –n, donde n es un número entero positivo, hace que la impresión se realice en n columnas.

edwar@localhost:~> pr -2 nombres.txt

2009-03-23 nombres.txt pagina 1

noddy	mickey
tom	asterix
jerry	tintin
donald	obelix

edwar@localhost:~>

El comando ha mostrado los 8 nombres en 2 columnas
Se puede notar una pantalla en blanco cuando se ejecuta lo anterior. El comando pr usa una longitud de página predefinida en la cual imprime los datos anteriores. Como el número de líneas es solo 4 en el ejemplo anterior, las líneas siguientes se toman como líneas en blanco. Si se incluyeran mas nombres en ese archivo, se podrá ver como se llena una página.

Fin de Uso de Comando 2.13

Uso de Comando 2.14

El comando **cut** se usa para mostrar partes seleccionadas, usualmente columnas o campos, de un archivo en la salida estándar. Esto es útil cuando solo se quiere ver una porción de un archivo.

Para entender este comando, se asumirá que las siguientes líneas están en un archivo, mislineas.txt

```
Esta es mi primera linea de texto
Esta es mi segunda linea  de texto
1 2 3 4 5 6 7
9 10 11 12 13 14
```

Sintaxis de cut: [opción] [archivo] cut imprime partes de un archivo en la salida estándar. Las opciones que se usan con cut son –f y –b. –f se usa para mostrar campos específicos y –b para mostrar bytes específicos. Se presentaran unos ejemplos para entender esto. La opción –d es para cambiar el delimitador. El delimitador es un solo carácter que separa cada columna en una fila. El delimitador por defecto es la tabulacion. Se puede cambiar el delimitador a un espacio o cualquier otro que se quiera.

```
edwar@localhost:~>  cut mislineas.txt
Esta es mi primera linea de texto
Esta es mi segunda linea  de texto
1 2 3 4 5 6 7
9 10 11 12 13 14
edwar@localhost:~>
```

En el ejemplo anterior, el comando cut imprime todas las líneas, dado que el delimitador por defecto para separar las columnas es la tabulacion y se tienen espacios en blanco como delimitador.
cut –f1 imprime la primera columna. Dado que no hay tabulaciones para separar las columnas, toda la fila será tratada como una columna.
Ahora se rectificara esto:

```
edwar@localhost:~>   cut –d ‘ ‘ –f1 mislineas.txt
Esta
Esta
1
9
edwar@localhost:~>
```

Se ha cambiado el delimitador a un espacio en blanco y así solo se ve la primera columna de cada fila. La opcion –d solo se usa con campos.

Para imprimir varias columnas, se usa el siguiente comando:

```
edwar@localhost:~>   cut –d ‘ ‘ –f1,3,5 mislineas.txt
Esta es mi linea
Esta es mi linea
1 3 5
9 11 13
edwar@localhost:~>
```

También se puede imprimir a nivel de bytes

```
edwar@localhost:~>   cut –b1 mislineas.txt
E
E
```

```
        1
        9
edwar@localhost:~>
```

En el ejemplo dado, se imprime el primer byte de cada fila, que solo es un carácter. Un carácter ocupa un byte en los sistemas Linux.

Ahora se vera como pueden imprimirse varios bytes de una sola fila, para todas las filas en un archivo. Recuerde, el comando cut trabaja sobre todo el archivo, por cada fila.

```
edwar@localhost:~>   cut –b1,6,12 mislineas.txt
        Eep
        Ees
        1
        911
edwar@localhost:~>
```

En el ejemplo, se ven impresos el primero, sexto y doceavo byte. La tercera linea de la salida solo contiene 1. Esto es por que el sexto y doceavo byte en la tercera linea son espacios en blanco. Note que no hay espacio entre b y los números subsecuentes.

Los ejemplos usando las opciones –f y –b muestran claramente la diferencia entre sus usos. Uno trata con un conjunto de caracteres (como parte de un campo o columna) y el otro con un solo carácter.

Fin de Uso de Comando 2.14

A continuación se discutirán brevemente el uso de otros comandos simples y se proporcionara además una explicación simple de cada uno. Estos son:

talk nombre _ ingreso.

El sistema Linux le permite hablar con otro usuario en el sistema, si el usuario esta conectado actualmente. El comando talk copia líneas de un usuario a otro cuando están en linea.

wall mensaje.

El comando wall envía el mensaje ingresado por un usuario a todos los usuarios actualmente conectados al sistema. Esto es útil normalmente cuando el administrador del sistema quiere compartir un mensaje con todos los que han ingresado. Un uso típico de esto, es cuando el sistema necesita tumbarse (shutdown) mientras los usuarios están conectados. El administrador del sistema envía un mensaje a todos los usuarios solicitándoles que salgan antes de apagar el sistema.

mail

El comando mail permite a los usuarios tanto enviar como recibir correos. El comando mail dirección permitirá al usuario enviar correo a la dirección mencionada luego del comando mail. Ingresar simplemente mail, sin ninguna dirección a continuación, permitirá a los usuarios leer los correos de su cuenta. Linux también proporciona otro programa llamado pine, que tiene una buena interfaz de usuario para ayudar al usuario a leer mejor el correo.

who

El comando who muestra todos los usuarios que están conectados actualmente en el sistema. Muestra el nombre de ingreso, el número de terminal y el momento en que ingresaron.

whoami

El comando whoami ayuda a encontrar quien esta conectado actualmente. En caso que se olvide el nombre del ingreso, este comando muestra el nombre de ingreso. Es útil para administradores de sistema, dado que les permite ingresar como diferentes usuarios para monitorear el sistema.

7. Tuberías (Pipes)

La redirección en Linux ayuda a conectar programas con archivos, mientras que las tuberías ayudan a conectar programas con otros programas. Las tuberías en linea es una característica donde la salida de un programa es enviada como entrada para otro programa. El carácter | (una barra vertical) representa una tubería.

Puede ser representado en forma de diagrama como se muestra en la figura 2.2

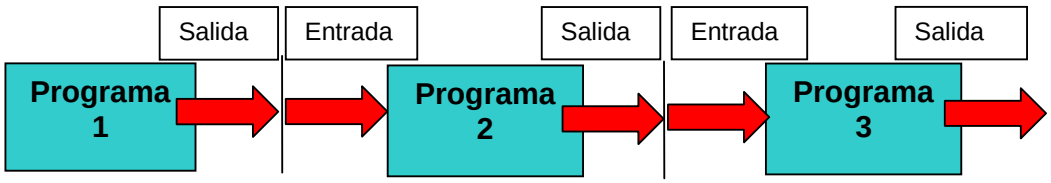

Figura 2.2 Tuberías en Linux

Uso de Comando 2.15

edwar@localhost:~> cat lineas.txt | grep esper | wc –l
3
edwar@localhost:~>

La salida cat es enviada como entrada a grep. El comando cat no procesa datos. Simplemente envía la entrada a la salida estándar. Pero antes que pueda mostrarse en la salida estándar, se encuentra el símbolo tubería (|), que ayuda al shell a entender que la salida debe ser enviada como entrada al siguiente comando en la tubería en linea.
El comando grep toma la entrada (y por lo tanto no se da como un archivo como argumento) y lo procesa buscando el patrón esper en cada linea, además envía la salida. Esta salida es pasada nuevamente a traves de la tubería a wc, que cuenta el numero de líneas y finalmente lo que se muestra es 3 (que es el numero de líneas totales que contienen el patrón esper, compárese con el ejemplo de uso de comando numero 12).
El uso de tubería presentado encuentra que existen tres líneas en el archivo lineas.txt que tienen el patrón esper en ellas.

edwar@localhost:~> head -5 lineas.txt | grep esper | sort

No se muestra salida alguna, dado que el patrón esper no existe en las cinco primeras líneas del archivo lineas.txt.

Fin de Uso de Comando 2.15

8. Filtros

Un filtro es un programa que lee datos de la entrada estándar, los procesa y envía los datos procesados a la salida estándar. La figura 2.3 ilustra un filtro.

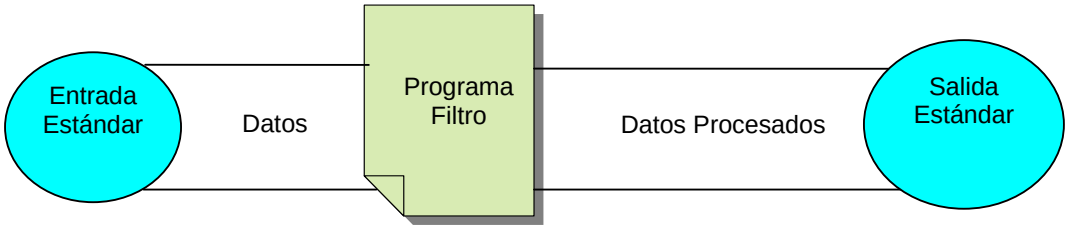

Figura 2.3 Filtros en Linux

El comando sort es un ejemplo de un filtro. Toma la entrada de la entrada estándar (ya sea a traves del teclado o un archivo), procesa los datos (los ordena en el orden que se requiere) y los envía a la salida estándar (ya sea la pantalla o un archivo, si se usa redirección).
Cualquier filtro puede ser usado en una tubería. A continuación se muestra un ejemplo:

cat nombres.txt | sort

Es lo mismo que sort nombres.txt.
Sin embargo, no todos los comandos en Linux son filtros. Por lo tanto, es importante notar que el lado derecho de una tubería puede no siempre ser un filtro. Por ejemplo, cal no es

un filtro y su uso en cat nombres.txt | cal no es un uso valido en tuberías y filtros
La salida que se vera será el calendario para el mes actual. En situaciones como estas, el último comando se ejecuta y todos los demás datos se pierden en la tubería. Cuando estos son incorrectos se hacen con tuberías y filtros en un programa, el error mismo puede ser difícil de identificar y corregir. Por lo tanto, se debe ser cuidadoso para asegurar que el uso en el lado derecho de una tubería sea un filtro correcto.
Se entenderán tuberías y filtros con otro ejemplo.
En una sección anterior se aprendió sobre head y tail. Ahora se vera como se puede imprimir las partes medias de un archivo. El comando head imprime el inicio de un archivo mientras que tail imprime las ultimas líneas. Se usara una combinación de ambos para imprimir algunas líneas del medio del archivo lineas.txt. Refiérase al contenido de este archivo dado en la sección anterior.
Suponga que quiere ver las líneas 4 5 6, esto es lo que se necesita hacer.
edwar@localhost:~> head -6 lineas.txt | tail -3
o
edwar@localhost:~> tail -8 lineas.txt | head -3
En el primer ejemplo, se obtienen las primeras seis líneas usando head y pasándolas a tail. Las ultimas tres líneas de las primeras seis líneas serán la 4 5 6
En el segundo caso, se obtienen la ultimas ocho líneas y luego se ven las tres primeras, que son la 4 5 6 líneas en el archivo. Se ha usado -8 dado que hay 11 líneas en el archivo. Se sabe que se puede encontrar el numero de líneas en un archivo usando el comando wc –l.

Unidad 3: Laboratorio del Sistema Linux

Objetivos de Aprendizaje

Al final de esta unidad Ud. será capaz de:

- Ingresar y salir del sistema Linux
- Usar comandos simples como cat, calendar, grep, head, etc
- Trabajar con el redireccionamiento en Linux
- Usar tuberías y filtros para obtener los resultados deseados del procesamiento de un archivo

Ejercicios de Laboratorio

1. Ingrese al sistema y cambie su contraseña. Salga y vuelva a entrar con la nueva contraseña. Realice el ejercicio dado, antes de empezar a trabajar en los ejercicios a continuación.

2. Usando el comando cat, cree dos archivos con los datos de los nombres y líneas (luego de Uso de Comando 6) dados en la Unidad 2: *El Sistema Linux.* Llámelos nombres.txt y lineas.txt

3. Muestre el contenido de nombres.txt

4. Encuentre si el patrón ‘y’ existe en las ultimas seis líneas de lineas.txt

5. Encuentre el numero de líneas en que la letra ‘z’ aparece en las primeras cuatro líneas de lineas.txt. (Lea la pregunta cuidadosamente antes de intentar responderla)

6. Capture el calendario para el mes de Diciembre de 2001 en un archivo llamado calendar.txt

   Agregue el calendario de Noviembre de 2001 a calendar.txt. El mes de Noviembre debe preceder al mes de Diciembre en calendar.txt

   Sugerencia: Puede tener que usar tres comandos para lograr esto.

Unidad 4: La Estructura de Archivos Linux

Objetivos de Aprendizaje

Al final de esta unidad Ud. será capaz de:

- Explicar como Linux organiza su estructura de Archivos
- Describir el uso de los comandos de archivos.
- Discutir los directorios y los comandos relacionados a directorios
- Explicar como se otorgan permisos a los archivos.

1. Introducción

Se discutió brevemente el sistema de archivos en Linux en la Unidad 1: *Fundamentos de Linux.* Ahora se aprenderá como los archivos están organizados en el sistema operativo Linux

Existen tanto una vista física como lógica del sistema de archivos. Los archivos se almacenan físicamente en el disco dentro de un sistema de archivos. Lógicamente, un sistema de archivos Linux puede ser visto como una estructura jerárquica (tipo arbol).

Se va a entender a continuación que son los archivos y directorios y los comandos relacionados con estos.

2. Archivos

Estas son algunas de las principales características de los archivos:

- Cada recurso en Linux es un archivo
- Un archivo es una secuencia de bytes
- Cada archivo tiene un nombre. Sin embargo, el uso de extensión en un nombre de archivo es opcional. Por lo tanto, archivodetexto, miarchivo.txt y miarchivo.doc son todos nombres validos. El .txt y .doc son extensiones de los nombres.
- Los archivos son almacenes para datos. Los datos pueden ser simple texto ASCII, documentos con formato, programas fuentes, archivos ejecutables o imágenes

El comando file puede usarse para determinar el tipo de archivo.

```
edwar@localhost :~> file lineas.txt
        lineas.txt: ASCII text
```

En el ejemplo anterior, se sabe que lineas.txt es un archivo ASCII y se puede determinar que el archivo es del tipo ASCII text (Texto ASCII)

3. Directorio

Un directorio es un almacén para archivos. El sistema Linux usa archivos y directorios para organizar los datos. Estas son algunas de las características de un directorio:

- Pueden guardar cualquier tipo de archivo
- Todos los directorios tienen un nombre, pero a diferencia de los archivos, no tienen extensiones.
- Un directorio Linux es un archivo especial que mantiene una lista de todos los archivos almacenados en el.
- Este esquema organizacional es poderoso, dado que un archivo en directorio puede ser otro directorio
- El directorio hijo es llamado un *subdirectorio* del directorio original. Así existe una estructura tipo arbol de directorios y archivos

Los directorios pueden ser comparados a ramas de un arbol, mientras que los archivos pueden ser comparados con las hojas. Como en los árboles, en Linux también existe una raíz (root), que almacena juntos todos los directorios y archivos.

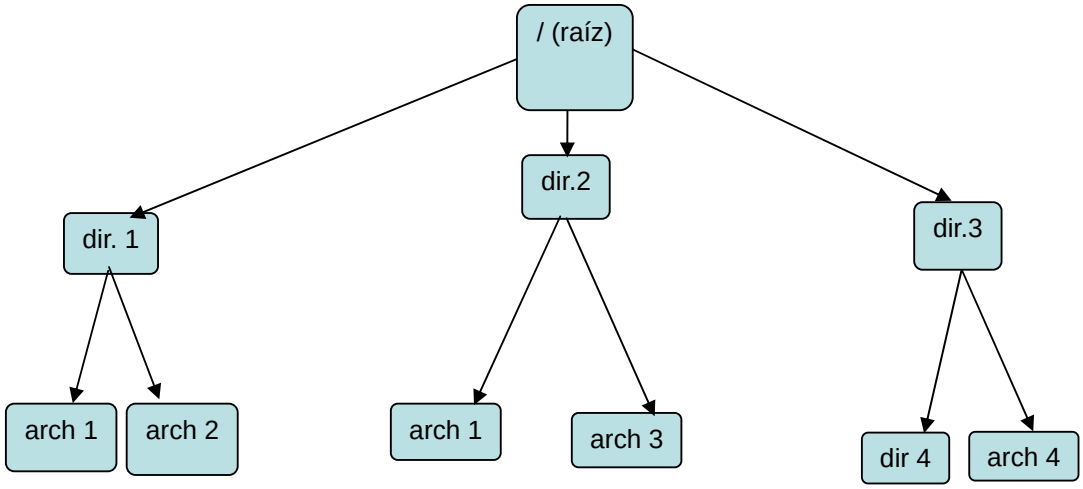

Figura 4.1: Estructura de Directorios en Linux

La raíz (root) en el sistema Linux es representada con una barra (*/*)
En el diagrama, dir1, dir2, dir3 son subdirectorios de root.
archivo1 y archivo2 son archivos en dir1, archivo1 archivo3 son archivos en dir2 y dir4 es un subdirectorio en dir3.
Los nombres de archivos y directorios deben ser únicos dentro del mismo subdirectorio sin embargo, pueden ser idénticos a lo largo de los subdirectorios. En la figura 4.1 archivo1 se encuentra en dir1 y dir2.

4. Inodo

Un archivo tiene varios componentes como se lista a continuación:

- Nombre
- Contenido
- Permisos
- Fecha de modificación

Un directorio es un tipo especial de archivo. Por lo tanto, en esta sección se usara el término de archivo para referirnos a archivo o directorio.
La información acerca de los permisos y fecha de modificación para un archivo es llamada información *administrativa*. La fecha del permiso es el instante de tiempo en que los permisos de lectura, escritura y ejecución fueron establecidos o modificados para el archivo. La fecha y hora de modificación se refiere al instante de tiempo en que el archivo fue cambiado. La información administrativa se almacena en una tabla llamada i-nodo o inodo. Cada archivo en Linux esta asociado a un inodo. Un inodo también es un archivo en Linux. Los tres tipos de tiempos (fecha-hora) de modificación que almacena un inodo son:

- La fecha y hora en que el contenido del archivo fue modificado por última vez. La modificación causa un cambio al contenido del archivo.
- La fecha y hora en que el archivo fue usado por última vez, esto es, leído o ejecutado. Este instante de tiempo se refiere al momento en que el archivo fue leído o ejecutado, sin causar ninguna modificación al archivo.
- La fecha y hora en que el inodo fue cambiado. Los inodos cambian cuando los permisos del archivo son alterados.

Cada inodo esta asociado a un i-numero único, que viene de número de identificación (identification number). En la Unidad 1: *Fundamentos de Linux* se aprendió acerca de los bloques inodo. Estos son bloques donde Linux almacena los inodos.
Se han visto las diferentes fechas almacenadas en un inodo. Aparte de esto, un inodo también mantiene la siguiente información:

- **Identificación del propietario del archivo**

Un individuo o grupo pueden ser propietarios de un archivo. Los detalles respecto a los usuarios que tienen derechos de acceso al archivo se mantienen en el inodo

- **Tipo del archivo**

En el cual se encuentra la información de si el tipo de archivo es regular o un directorio. También se mantiene la información de si el archivo es tipo carácter o de bloque especial.

- **Numero de enlaces al archivo**

Un archivo puede residir físicamente en un directorio y tener cierto numero de enlaces a el desde otra secciones del sistema de archivos. El inodo mantiene el número de nombres que tiene el archivo en la jerarquía de directorios.

- **Tamaño del archivo**

Incluye el almacenamiento de información acerca del tamaño del archivo en bytes

- **Lista de direcciones de disco para el archivo**

La vista lógica de un archivo es que el contenido es almacenado en un flujo secuencial de byte. El kernel típicamente almacena el contenido en bloques de disco no contiguos. El inodo mantiene la lista de las direcciones en el disco donde reside cada parte del archivo.

La Figura 4.2 muestra un ejemplo de un inodo

Propietario/Grupo
Tipo de Archivo
Tamaño del Archivo
Permisos del Archivo (rwxrwxrwx)
Fecha/Hora de: Modificación, Acceso y Cambio
Cantidad de Enlaces
Banderas Adicionales (ACL, EXT2, FLAGS)
Punteros de Bloques de Datos

Figura 4.2: Inodo Ejemplo

5. Comandos de Directorio

Ya se conoce como crear un archivo. Ahora, para colocar los archivos en diferentes directorios, estos directorios tienen que ser creados. Cuando se instala Linux, se crea el directorio / (raíz) y muchos otros subdirectorios para archivos relacionados al sistema operativo. Sin embrago, cuando el usuario ingresa, estará en su directorio de inicio. Puede crear subdirectorios y archivos en solo sus respectivos directorios home.

A continuación se muestra como utilizar comandos relacionados con directorios. Estos comandos permiten crear un directorio, mover archivos a un directorio, eliminar un directorio, encontrar la ruta del directorio de trabajo actual y listar el contenido de un directorio.

Se discutirán los siguientes comandos:

mkdir
cd
rmdir
pwd
ls

Uso de Comando 4.1

Para organizar mejor el especio en disco, Linux permite crear directorios. El comando **mkdir** se usa para crear un directorio.

Sintaxis de mkdir: mkdir [opcion] directorio

edwar@localhost:~> mkdir programas

edwar@localhost:~> mkdir documentos proyectos

El primer mkdir crea un directorio llamado programas en el directorio home del usuario
El segundo mkdir crea dos directorios mas llamados documentos y proyectos en el mismo home
Cuando el administrador del sistema crea una cuenta para el usuario (edwar) se crea automáticamente un directorio llamado edwar en el sistema, en el directorio home. El directorio home mismo es subdirectorio de raíz (/)
Los directorios programas, documentos y proyectos son subdirectorios del directorio edwar. El comando mkdir puede tomar uno o mas parámetros
Si el directorio aun no existe, se crea un directorio con el nombre especificado. Caso contrario se mostrara un mensaje de error adecuado, como se muestra a continuación:

edwar@localhost:~> mkdir programas
mkdir: no se puede crear el directorio <<programas>>: El fichero ya existe
edwar@localhost:~>

Fin de Uso de Comando 4.1

La estructura de arbol resultante después de crear los tres subdirectorios usando el comando mkdir, se muestra en la figura 4.3.

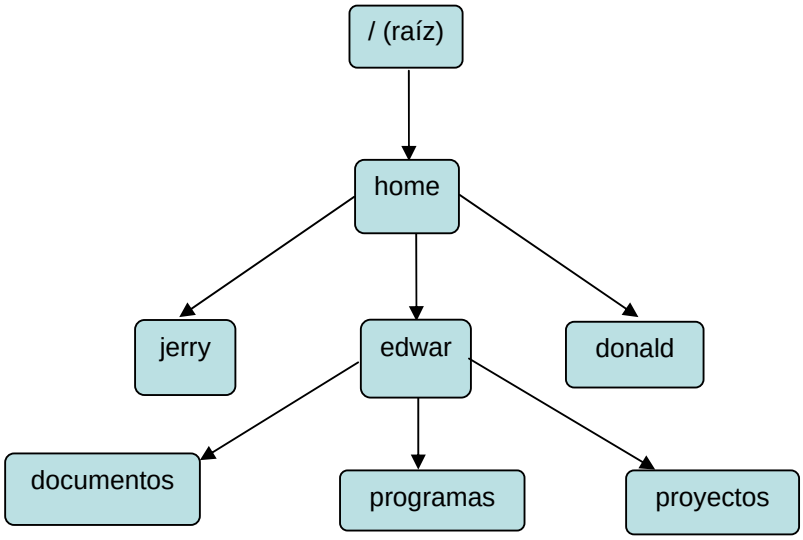

Figura 4.3 Estructura de Directorios luego de crear Subdirectorios

Uso de Comando 4.2

Una vez que se crea un directorio, se tiene que ir a dicho directorio para crear archivos y otros subdirectorios. El comando mkdir simplemente lo crea. Para poder trabajar en el directorio recién creado, y crear archivos y subdirectorios, se debe convertir ese directorio en el directorio actual de trabajo. Esto se hace usando el comando **cd**, siglas de *change directory* (cambiar directorio)

Sintaxis de cd: cd directorio

edwar@localhost:~> cd programas
edwar@localhost:~/programas>

El usuario edwar ahora esta en el directorio programas. Considere el nuevo prompt:
edwar@localhost:~/programas>
Esto muestra que el directorio en el que se esta, es de hecho donde se quiere estar.
Existen dos símbolos especiales estos son . y .., que representan directorios específicos.
El . (punto) representa el directorio actual y .. (dos puntos) representa el directorio padre.
Sea este ejemplo:

edwar@localhost:~/programas>cd..
edwar@localhost:~>

Para el subdirectorio programas, el directorio padre es edwar. Por lo que cuando se ingresa cd .. lleva al /home/edwar.

Fin de Uso de Comando 4.2

Uso de Comando 4.3

El comando **rmdir** se usa para eliminar un directorio existente del sistema de archivos.

Sintaxis de rmdir: rmdir [opcion] directorio

```
edwar@localhost:~> rmdir proyectos
edwar@localhost:~>
```

Si intenta eliminar el directorio proyectos. Si el directorio proyectos esta vacío se eliminara. En caso contrario, mostrara el siguiente mensaje

```
rmdir: proyectos/: El directorio no esta vacío.
```

Fin de Uso de Comando 4.3

Después de desplazarse por el sistema de archivos, el usuario puede desorientarse. Para permitir al usuario determinar en que directorio esta actualmente, Linux proporciona un comando simple pwd. Este comando muestra el nombre del archivo completo del directorio de trabajo actual. El comando **pwd** no es un comando relacionado a directorios, pero proporciona información acerca del directorio de trabajo en que se esta ubicado.

Uso de Comando 4.4

Sintaxis de pwd: pwd [opcion]

```
edwar@localhost:~> pwd
/home/edwar
edwar@localhost:~> cd documentos
edwar@localhost:~/documentos>pwd
/home/edwar/documentos
edwar@localhost:~/documentos>
```

El nombre completo del directorio de trabajo actual de trabajo edwar es /home/edwar. Cuando se cambia el directorio a documentos, el nombre completo es /home/edwar/documentos.

Fin de Uso de Comando 4.4

El contenido de un archivo de texto regular, puede visualizarse usando un editor o procesador de palabras, mientras que el contenido de un directorio puede visualizarse usando el comando ls. El contenido de un directorio son archivos y otros subdirectorios creados en el.

Uso de Comando 4.5

La estructura de directorios dada en la Figura 4.4, forma la base para la explicación del comando **ls**.

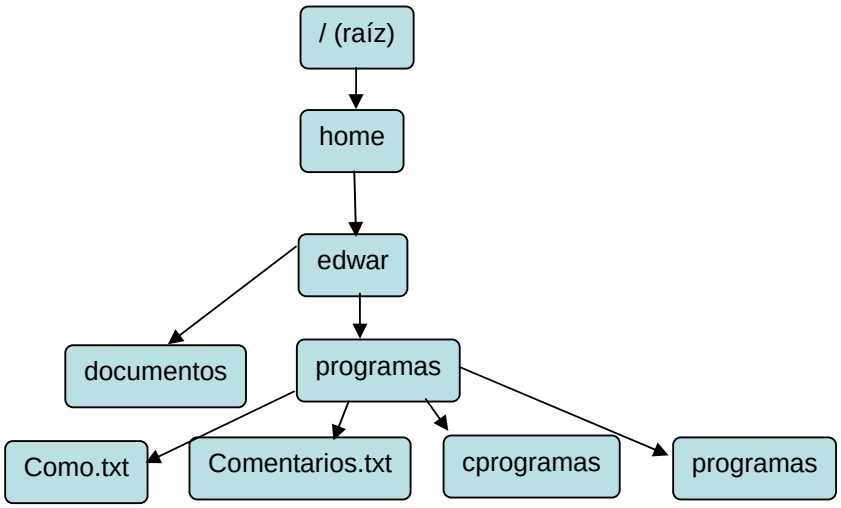

Figura 4.4 Ejemplo de un Estructura de Directorios

Sintaxis de ls: ls [opcion] [archivo]

Se tomara el ejemplo de la estructura de directorios mostrada en la Figura 4.4. Todos los directorios están subrayados y el nombre del archivo completo del directorio de trabajo actual es:
/home/edwar/programas.

```
edwar@localhost:~/programas> ls
        comentarios.txt     cprogramas  como.txt      programas
edwar@localhost:~/programas>
```

El contenido del directorio de programas se lista a continuación. Esta es la forma mas simple en que el comando ls puede ser usado.

```
edwar@localhost:~/programas> ls –l
        total
        -rw-r—r--    1 edwar users      2     2009-03-19  15:00  comentarios.txt
        -rw-r—r--    1 edwar users            2009-03-19  15:00  como.txt
        drwxr-xr-x   2 edwar users      48    2009-03-19  15:00  cprogramas
        drwxr-xr-x   2 edwar users            2009-03-19  15:00  programas
edwar@localhost:~/programas>
total 16
```

La forma mas útil de ls es con la opcion –l ,ls –l da un listado 'largo' que proporciona más información acerca de cada archivo. En el resultado, 'total 16' indica cuantos bloques son usados por los archivos en el disco. La cadena "-rw-r-r—"informa acerca de los permisos sobre los archivos. Se apredendera mas acerca de esto en una sección posterior. '1' es el numero de enlaces al archivo 'edwar' es el dueño del archivo. '4096' es el numero de caracteres en el archivo. Esto es seguido por la fecha, hora de creación y nombre del archivo. Si el archivo es un directorio, entonces se vera una d en la cadena que representa los permisos sobre ese directorio
ls –i mostrara los archivos con sus números de inodos

Fin de Uso de Comando 4.5

A continuación se presentan todos los usos validos de los comandos relacionados a directorios.

edwar@localhost:~> cd/home/jerry (Esto es valido si edwar tiene permiso para cambiarse al directorio jerry.)

edwar@localhost:~> mkdir documentos/docespeciales. (Esto crea un subdirectorio llamado docespeciales en el subdirectorio documentos)

edwar@localhost:~> rmdir documentos/docespeciales (Esto elimina el subdirectorio docespeciales, si esta vacío)

edwar@localhost:~> ls/home/edwar/documentos

edwar@localhost:~> ls documentos.

En la primera forma de uso, se da la ruta desde la raíz. En la segunda, solo se da el nombre del directorio de cuyo listado se esta interesado. Para poder hacer esto, documentos debe ser un subdirectorio del directorio actual de trabajo.

6. Comandos de Archivo

A continuación se presentan los comandos que trabajan específicamente con archivos. Para trabajar con archivos el primer paso es crear un archivo. Se aprendió a crear un archivo con el comando cat.

edwar@localhost :~> cat > miarchivo.txt.

Hay otras formas de crear un archivo a traves de editores. Un editor es un programa que permite la creación y modificación de un archivo.

vi es un editor disponible en los sistemas Linux, es un editor visual. Se aprenderá sobre este editor en la *Unidad 6 – El Editor vi*.

Otro editor es ed, es un editor de linea. Los editores de linea son aquellos que trabajan con una sola linea a la vez.

El comando cat simplemente permite crear un archivo a los usuarios. No les deja modificar el contenido del archivo. Por otro lado, ed permite al usuario crear un archivo y modificar su contenido.
Se discutirán brevemente algunas características importantes del editor ed.

Uso de Comando 4.6

Sintaxis de ed: ed [archivo]

```
edwar@localhost:~>  ed
a
Este archivo fue creado usando el editor ed.
Creado el 30 Marzo 2009
A las 17:05
w miarchivo.txt
86
q
edwar@localhost:~>
```

Si se ejecuta el editor sin un nombre de archivo, este asume que es un nuevo archivo. El nombre de un archivo puede ser especificado dentro del editor cuando se graben las líneas ingresadas. Para ingresar líneas existen dos comandos en ed **a** e **i**
El comando **a** viene de append (añadir) y el comando **i** viene de insert (insertar).
En el ejemplo anterior, se encuentran tres líneas añadidas luego del comando **a.** los comandos del ed son simples y ellos deben estar solos en una linea. Para indicar la finalización del ingreso, se utiliza el "." (otro comando). Esto informa a ed que no se esta en el modo de edición, sino que ahora se esta en el modo de comandos.
El editor ed se inicia en el modo de comandos.
Las líneas que se ingresaron solo están en la memoria de la computadora. Para escribir las líneas en un archivo del disco se usa el comando **w** seguido por el nombre del archivo. El nombre del archivo se ingresa solo la primera vez. Posteriormente, se puede ingresar simplemente el comando **w**. Las líneas se guardaran en el mismo archivo. El comando **w** escribe las líneas a un archivo en el disco, en este caso al archivo llamado miarchivo.txt y muestra el numero de caracteres escritos que es de 86.
El comando **q** termina el programa. Si se usa **q** sin haber escrito las líneas en el archivo, ed mostrara ? Esto es para informar que puede que no se estén haciendo las cosas correctamente.

```
edwar@localhost:~>  cat miarchivo.txt
Este archivo fue creado usando el editor ed.
Creado el 30 Marzo 2009
A las 1732
edwar@localhost:~>
```

Usando cat se puede ver que las tres líneas ingresadas a traves de ed están correctas

```
edwar@localhost :~> ed miarchivo.txt
        86
        a
        Añadiendo otra linea.
        Esta linea fue añadida 1740
        w
        141
        q
edwar@localhost:~>
```

Esta vez, se usa ed con el nombre del archivo a continuación. Se abre el archivo si existe, en caso contrario se muestra un mensaje de error y espera por otra entrada del usuario. Si el archivo no existe, se puede continuar y usar el ed como se hizo previamente (para crear el archivo). El programa no terminara.

Si el archivo existe, se mostrara el número de caracteres que contiene y se esperara por la entrada del usuario. Se han agregado dos líneas mas y luego fueron grabadas. Note que no se ingresa el nombre del archivo después del comando **w**. El archivo miarchivo.txt con las líneas añadidas se muestra a continuación usando el comando cat.

```
edwar@localhost:~> cat miarchivo.txt
        Este archivo fue creado usando el editor ed.
        Creado el 30 Marzo 2009
        A las 1732
        Añadiendo otra linea.
        Esta linea fue añadida 1740
edwar@localhost:~>
```

Fin de Uso de Comando 4.6

Habiendo aprendido otro metodo para crear archivos, se van a estudiar a continuación algunos comandos específicos relacionados a archivos. Estos comandos permiten copiar de un archivo a otro, cambiar el nombre, eliminar y crear enlaces a archivos.

- cp
- mv
- rm

Uso de Comando 4.7

El comando **cp** copia el contenido de un archivo a otro.

Sintaxis de cp: [opcion] origen destino

```
edwar@localhost:~> cp miarchivo.txt nuevoarchivo.txt
edwar@localhost:~>
```

El contenido de miarchivo.txt se copia a nuevoarchivo.txt. El comando cp sobrescribe el archivo de destino. Vea el siguiente ejemplo:

```
edwar@localhost:~> cp miarchivo.txt documentos/nuevoarchivo.txt
```

miarchivo.txt se copia desde el directorio edwar al directorio documentos como nuevoarchivo.txt.

A veces el nombre destino puede no especificarse como el siguiente ejemplo:

```
edwar@localhost:~> cp miarchivo.txt documentos.
edwar@localhost:~>
```

Aquí miarchivo.txt se copia al directorio documentos con el mismo nombre miarchivo.txt
El sistema reconoce que documentos es un directorio y por ello realiza la copia de esa manera.

Fin de Uso de Comando 4.7

Cuando se copia un archivo, se crean dos archivos físicos. El comando mv se usa para renombrar un archivo existente. El comando da un nuevo nombre al archivo especificado, pero sigue existiendo solo una copia del archivo.

El comando **mv** también puede usarse para mover archivos de un directorio a otro

Uso de Comando 4.8

Sintaxis de mv: mv [opcion] origen destino

```
edwar@localhost:~> mv miarchivo.txt nuevoarchivo.txt
edwar@localhost:~>
```

El comando anterior cambia el nombre de miarchivo.txt a nuevoarchivo.txt. El comando mv puede usarse también para mover archivos como se muestra en el siguiente ejemplo:

```
edwar@localhost:~> mv miarchivo.txt  documentos.
```

Aquí miarchivo.txt se mueve al directorio documentos. El archivo ya no existirá en el directorio edwar.

Fin de Uso de Comando 4.8

Uso de Comando 4.9

Un archivo puede ser eliminado usando el comando **rm**, siglas en ingles de la palabra remove (eliminar).

Sintaxis de rm: rm [opcion] archivo.

edwar@localhost:~> rm miarchivo.txt

Esto elimina miarchivo.txt del directorio edwar. Se puede eliminar archivos desde cualquier parte del sistema, siempre que se tengan los permisos necesarios para hacerlo. Se aprenderá sobre esto en la siguiente sección. Si hay más de un enlace al archivo, rm solo elimina el enlace actual. El archivo se elimina completamente del sistema cuando se eliminan todos los enlaces.

El comando rm –r se puede usar para eliminar el contenido de los subdirectorios recursivamente. Así, si el argumento de rm –r en el ejemplo mostrado, es un directorio y no archivo

Si embargo, los usuarios deben ser muy cuidadosos al usar el comando con –r. El comando con la opcion –r opera de la siguiente forma:

Si el argumento es un archivo simplemente lo elimina. Si el argumento es un directorio, entonces simplemente elimina el contexto del directorio y el directorio mismo sin ningún aviso.

Fin de Uso de Comando 4.9

Algunos comandos como pr, rm, ls y muchos otros pueden usar comodines en los nombres de los archivos. Dos de los mas usados son * y ?, donde * significa “cualquier cadena de caracteres” y ? equivale a ‘un carácter cualquiera’. Vea el siguiente ejemplo:

edwar@localhost:~> ls *.txt

Este comando mostrara todos los archivos con la extensión .txt

edwar@localhost:~> ls chap*

El comando anterior, por otro lado, mostrara los archivos que tienen los cuatro primeros caracteres como chap. La cadena chap puede estar seguida de cero o más caracteres.

edwar@localhost:~> ls chap?

Este ejemplo mostrara todos los archivos que tienen los cuatro primeros caracteres como chap, pero seguidos por un solo carácter.

7. Permisos de Archivo

Cada archivo en Linux tiene un conjunto de permisos asociados. Estos permisos le dicen al sistema Linux quien puede hacer que con el archivo. Los permisos ayudan a asegurar que los datos almacenados en el archivo no sean dañados por otros.

Linux divide a los usuarios en tres categorías:

- Usuario
- Miembro de un Grupo de Usuarios.
- Otros usuarios

Los grupos de usuario contienen dos o más usuarios pertenecientes a un grupo particular, tal como las personas que trabajan en el mismo proyecto. Normalmente, cuando el administrador del sistema crea cuentas para los usuarios, ellos son asociados a un grupo.

Para cada categoría, el sistema mantiene tres tipos de permiso: lectura, escritura y ejecución. De esta manera, hay 3 categorías y 3 permisos para categoría, resultando un total de 9 permisos que pueden establecerse para un archivo. Si solo se otorga permiso de lectura a los usuarios sobre un archivo, entonces ellos solo pueden leer el contenido del archivo, pero no pueden realizar ninguna modificación en el.

Linux asigna diferentes valores para estos tres tipos de permisos:

- 4 para lectura
- 2 para escritura
- 1 para ejecución

El sistema usa el sistema de numeración octal para establecer los permisos. Este sistema también se llama sistema de base-8. Tiene 8 dígitos validos, 0, 1, 2, 3, 4, 5, 6, 7. Un

numero octal valido es una combinación de uno o mas dígitos de cualquiera de los dígitos validos de 0 a 7. El numero 456 es un numero octal valido, pero 458 no. Es importante notar que 456 en octal NO es lo mismo que 456 en decimal.
Con los ocho dígitos, ocho combinaciones diferentes son posibles para cada categoría.
0 - - - ni lectura, ni escritura, ni ejecución (permiso sin sentido)
1 - - x solo ejecución
2 -w -- solo escritura
3 – wr escritura y ejecución
4 r - - solo lectura
5 r – x lectura y ejecución
6 rw- lectura y escritura
7 rwx lectura, escritura y ejecución
r indica lectura (read), w indica escritura (write) y x indica ejecución (execute).
Se presenta un ejemplo. El administrador del sistema quiere dar el siguiente permiso:

- Todos los permisos al usuario
- Permiso de lectura y ejecución para el grupo al que pertenece el usuario
- Permiso de solo lectura par los otros

Los permisos en octal se escribirán como sigue:

- 7 para el usuario
- 5 para el grupo de usuarios
- 4 para los otros.

De este modo el permiso se escribe 754
Linux proporciona el comando **chmod** (siglas en ingles de change mode) para asignar permisos a los archivos y directorios, además de emplearse para cambiar los permisos ya otorgados.

Uso de Comando 4.10

Sintaxis de chmod: chmod [opcion] permiso archivo
edwar@localhost :~> chmod 754 miarchivo.txt
edwar@localhost :~>

El comando anterior otorga permisos de lectura, escritura y ejecución al Usuario; lectura y ejecución al Grupo de Usuarios y solo lectura para los otros.
Pero no tiene sentido dar permisos de ejecución a un archivo de texto. Los archivos de texto solo pueden ser leídos y modificados. Ejecutar un archivo es correr un programa o comando. Asuma que se ha escrito un mifecha.c y que se ha creado un programa ejecutable mifecha.out después de compilar y enlazar. No se puede usar el siguiente comando:

edwar@localhost:~> mifecha.c

Sin embargo puede usarse el comando dado a continuación:

edwar@localhost:~> mifecha.out

mifecha.c es un archivo de texto y solo puede ser visualizado a traves del comando cat o un editor. Ahora, para el archivo mifecha.out, se puede asignar el siguiente permiso:

edwar@localhost:~> chmod 710 mifecha.out.

Tradicionalmente, los archivos ejecutables en Linux tienen la extensión .out. Al dar el comando anterior, se ha asignado todos los permisos al usuario. El usuario, normalmente tiene todos los permisos por defecto para un archivo (6 para los archivos de texto y 7 para los ejecutables). Se ha asignado permiso de ejecución para el grupo de usuarios. Se requiere que el grupo pueda ejecutar el programa. Pero no se ha asignado permisos para los otros. ningún otro usuario en el sistema podrá ejecutar este programa. De esta manera se asegura que sus archivos no sean estropeados por otros.
Nota: Solo el dueño de un archivo puede asignar sus permisos.

Fin de Uso de Comando 4.10

Ejecutar un archivo es directo. Pero ¿Qué se quiere decir con ejecutar un directorio?. Si no se asigna un permiso de ejecución a un directorio, los usuarios no podrán hacer uso del nombre del directorio dentro de una ruta.
Sea la siguiente sentencia:
edwar@localhost:~> chmod 750 documentos.
Esto asigna todos los permisos para el Usuario, lectura y ejecución para el grupo de usuarios y ningún permiso para los otros sobre el directorio *documentos*. Un Usuario perteneciente a la categoría otros, no podrá hacer lo siguiente (asumiendo que jerry es un Usuario de la categoría otros)
jerry@localhost:~> ls /home/edwar/documentos.
Es posible que edwar tenga todos los permisos para los otros. Pero como el directorio documentos esta protegido, el Usuario jerry no puede listar el contenido del directorio documentos perteneciente a edwar.
Los permisos de lectura y escritura en los directorios son simples. Leer es simplemente intentar ver el contenido del directorio. Escribir es la posibilidad de copiar, crear o modificar documentos en el directorio.
Sea el siguiente ejemplo:
edwar@localhost:~> chmod 751 documentos
Aunque se ha dado el permiso de ejecución a los otros sobre *documentos*, el usuario jerry no podrá listar el contenido del directorio *documentos*, pues no tiene permiso de de lectura. Sin embargo, el podrá usar el nombre del directorio en una ruta.
Sin permiso de escritura, el usuario no podrá llevar a cabo las siguientes tareas:

- Copiar un archivo dentro del directorio
- Crear un nuevo archivo en el directorio

Cuando se crea o copia un archivo en un directorio, el sistema debe escribir en el inodo del archivo directorio, el cual no tiene permiso de escritura.
Similarmente, si a un directorio solo se le dan permisos de lectura y ejecución, el usuario no puede copiar archivos de otros directorios en ese directorio, dado que el directorio no tiene permiso de escritura.
Solo la práctica en un sistema operativo Linux le ayudara a entender mejor las combinaciones de permisos y sus efectos.

Unidad 5: Laboratorio de Estructura de Archivos en Linux

Objetivos de Aprendizaje

Al final de esta unida Ud. será capaz de:

- Crear una estructura de directorios.
- Usar los diferentes comandos relacionados a archivos y directorios
- Establecer permisos sobre archivos.

Ejercicios de Laboratorio

1) En su directorio “home”, crear los siguientes directorios.
   linux
   cprogramas
   datastructures
   documentos
   En linux crear
   - proyectos
   - asignaciones

   En asignaciones de linux crear
   - unit1
   - unit2

   En cprogramas crear
   - proyectos
   - asignaciones

2) Usando ed crear un archivo llamado miprimerarchivo.txt y añada la pregunta anterior en el archivo
3) ¿En cual directorio fue creado miprimerarchivo.txt y por que?
4) Ingrese a proyectos dentro de linux (desde su ubicación actual) y cree un archivo, usando cat. Nombre el archivo como simple.txt
5) Copiar el contenido de simple.txt con el mismo nombre en el directorio documentos. Haga esto dos veces. Primero realice la copia estando en el directorio linux. Para la segunda vaya al directorio documentos y copie nuevamente. Esta vez coloque otro nombre al archivo, nuevosimple.txt
6) Elimine simple.txt
7) En el directorio documentos, ejecute ls y vea el listado completo de los archivos en este directorio. Asigne permisos a nuevosimple.txt de tal manera que el dueño tenga permiso de lectura y escritura, el grupo de usuarios tenga solo lectura y los otros no tengan ningún permiso.
8) Finalmente, intente ejecutar todos los comandos aprendidos en las dos unidades anteriores. Revise las páginas man para las opciones de los comandos y practíquelos.

Nota: Algunas de las preguntas en el ejercicio pueden depender de las preguntas anteriores. Por lo tanto, siga las instrucciones en cada pregunta cuidadosamente.

Unidad 6: El Editor vi

Objetivos de Aprendizaje

Al final de esta unidad Ud. será capaz de:

- Discutir como trabaja el editor visual vi en Linux
- Describir las diferentes categorías de comandos
- Explicar el uso de comandos para editar archivos

1. Introducción

Los editores se usan principalmente para crear documentos. Linux ofrece varios editores, algunos de ellos son: vi, emacs, ed y ex. Los editores como ed y ex se conocen como editores en 'linea'. Mientras que ed permite realizar la inserción, eliminación, modificación, etc linea por linea vi ofrece una facilidad de pantalla completa para crear y editar documentos. Se ha discutido en detalle acerca de ed y sus capacidades, en la *Unidad 4 – La Estructura de Archivos Linux*. En esta unidad, se discutirá el editor vi.

El termino 'vi' viene de 'visual editor', vi (en ingles 'vee eye') es el único editor que se encuentra en casi toda instalación Unix/Linux. Fue escrito originalmente en la Universidad de California en Berkeley y sus versiones ahora pueden encontrarse en casi todas las ediciones de proveedores de Linux, vi empezó como un editor de linea ex y evoluciono a lo largo de los años hasta convertirse en un poderoso editor visual, ex ahora existe en forma separada y es un modo de edición especial de vi. A traves de vi se pueden iniciar los comandos ex.

vi proporciona un conjunto de comandos para la inserción, eliminación y modificación de texto. Usando expresiones regulares se pueden realizar poderosas búsquedas y reemplazos de texto.

2. Modos en vi

vi proporciona tres modos: modo de comando, modo de entrada y modo ex. Cuando se invoca el programa vi, se esta en el modo comando. En este modo solo pueden usarse comandos vi validos. En el modo de entrada, se puede ingresar, eliminar y modificar texto. Mientras se esta en modo de entrada, si se quiere ingresar un comando, se puede regresar al modo de comando presionando la tecla escape <Esc>. Una vez ahí, se puede ingresar cualquiera de los comandos vi. Del modo de comando, se puede ir a los modos de entrada o ex. Normalmente, la mayoría de los comandos para grabar usados en vi son comandos ex.

La Figura 6.1 muestra la relación entre los tres modos

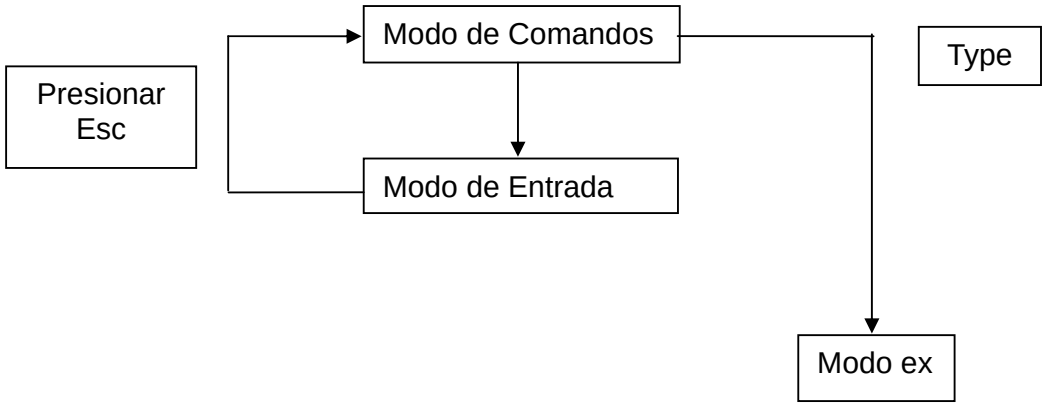

Figura 6.1: Modos del Editor vi

Las posibilidades de navegación de un modo a otro son:

- Del modo de comando al modo de entrada
- Del modo de comando al modo ex presionando la tecla dos puntos (:).
- Del modo de entrada al modo de comando presionando la tecla Esc.
- Del modo de entrada al modo ex, solo posible a traves del modo de comando

Una vez que se invoca un comando de ex presionando : (dos puntos), el editor regresa al modo de comandos.

3. Fundamentos de vi

Un editor vi puede abrirse ingresando vi seguido por un nombre de archivo como se muestra en el comando dado a continuación:
edwar@localhost:~> vi poema.txt.
Aparece la ventana del editor vi en pantalla.
El editor vi muestra el nombre del archivo en la parte inferior de la ventana. Se muestra [Fichero nuevo] porque es un archivo recién creado. Por defecto aparece una (-) en el lado izquierdo de la ventana del editor
vi estará en modo de comandos al ser invocado.
A continuación se explicara como puede escribirse un texto en el editor vi.
Se usaran líneas de texto de ejemplo para ilustrar el uso de los comandos. En esta unidad, se aprenderá acerca de los comandos que realizan las siguientes operaciones en un archivo:

- Insertar texto
- Guardar texto
- Eliminar texto
- Movimiento de pantalla
- Modificar texto
- Copiar y pegar texto
- Cortar y pegar texto
- Reemplazo de texto

El patrón adoptado en esta unidad para explicar los tres primeros comandos importantes que son insertar, guardar y eliminar es como sigue:

- Una breve introducción al tema.
- Texto de ejemplo para ilustrar comandos
- Una tabla, que muestra los comandos, propósito de los comandos y ejemplos usando el texto del ejemplo.
- Explicar los diferentes comandos tomando el poema ejemplo dado a continuación

Ejemplo de Poema:

I wanna brek free
From the sackles of this society,
From the rules and regulations governing it,
From its prevailing age-old.

Nota: hay palabras incorrectas en la primera (brek) y la segunda (sackles) líneas. Esto se hace a propósito para entender el uso de los comandos de eliminación de texto mas adelante en la unidad.

4. Inserción de Texto

En esta sección, se entenderá como ingresar texto en el editor vi. No se puede ingresar texto directamente en la ventana del editor vi, dado que el texto no aparecerá en la pantalla. A fin de iniciar el ingreso de texto en este editor, se tiene que ingresar uno de los *comandos de inserción*.

Texto Ejemplo:

mickeymouse

La tabla 6.1 ilustra los diferentes comandos de inserción y su propósito. Se usara ‘key’ como texto a insertar.

Comando	propósito	Ejemplo
I	Inserta a la izquierda del cursor	Si el cursor esta en ‘y’ entonces inserta el texto a la izquierda de ‘y’ Resultado: micke**key**ymouse
A	Inserta a la derecha del cursor	Si el cursor esta en ‘y’, entonces inserta el texto a la derecha de ‘y’ Resultado: mickey**key**mouse
I	Inserta al inicio de la linea	Inserta el texto a la izquierda de la primera letra ‘m’ de la palabra ‘mickeymouse’, sin importar la posición del cursor. Resultado: **key**mickeymouse
A	Inserta al final de la linea	Inserta el texto a la izquierda de la primera letra ‘e’ de la palabra ‘mickeymouse’, sin importar la posición del cursor. Resultado: mickeymouse**key**
O	Abre una nueva linea y agrega texto debajo de la linea actual	Ayuda a escribir una nueva linea debajo de ‘mickeymouse’ Resultado: mickeymouse <linea en blanco abierta para insertar texto>
O	Abre una nueva linea y agrega texto encima de la linea actual	Ayuda a escribir una nueva linea encima de ‘mickeymouse’ Resultado: <linea en blanco abierta para insertar texto> mickeymouse

Figura 6.1: Tabla de Comandos de Inserción vi

A continuación se aprenderá como usar los comandos de inserción de vi listados en la tabla 6.1, usando el poema de ejemplo. Cuando se ingresa la letra i, el estado INSERT aparece en la parte inferior de la ventana.

El comando i, cuando el editor vi se abre por primera vez, permite ingresar texto cambiando del modo de comando al modo de inserción. Ahora, se ingresara el poema y luego se regresara al modo comando presionando la tecla <Esc>.

Al terminar el ingreso inicial de las cuatro líneas del poema, el cursor estará posicionado bajo el final de la última linea. Asuma que ahora se quiere agregar una palabra al poema. Se ingresara el comando A y luego se ingresa ‘prototype’. La palabra se agregara al final como se muestra a continuación:

I wanna brek free
From the sackles of this society,
From the rules and regulations governing it,
From its prevailing age-old. Prototype.

Si se quiere ingresar la palabra ‘prototype’ al inicio de la linea, se usara el comando I.

A continuación se aprenderá a corregir el error de la palabra ‘brek’ en el poema. Se puede usar cualquier comando de inserción para la corrección de texto.

Dependiendo de que palabra se necesite corregir, se puede usar a, i, A o I.

Se debe agregar una ‘a’ para cambiar ‘brek’ por ‘break’. Usando las teclas del cursor para moverse por la pantalla, se posicionara el cursor bajo la letra ‘e’ de ‘brek’ y se ingresara a (el comando para inserción). El editor vi ahora esta en modo de inserción.

Se ingresara la letra ‘a’.

La otra forma de hacer esta corrección es mover el cursor hasta ‘k’, presionar el comando ‘i’ e ingresar ‘a’.

El poema corregido se vera como se muestra a continuación:

I wanna break free
From the sackles of this society,
From the rules and regulations governing it,
From its prevailing age-old. Prototype.

A continuación se va a aprender acerca de cómo guardar el trabajo que se ha creado.

5. Guardar un Archivo

Usando los comandos de inserción, se puede agregar texto y hacer modificaciones al texto. Para guardar en el almacenamiento secundario, se necesita usar los comandos para guardar disponibles en vi, a traves de ex. La tabla 6.2 muestra un conjunto de comandos vi disponibles para guardar un archivo. Los comandos empleados para guardar, mostrados en la tabla son comandos para guardar de ex, que son usados por vi.

Comando	propósito
w	Guardar y continuar el trabajo
wq	Guardar y salir de vi
q!	Salir sin guardar cambios

Tabla 6.2: Comandos para Guardar de vi

En el ejemplo, para guardar el poema, asumiendo que se esta en el modo de inserción, se debe cambiar al modo de comando. Cuando se cambia al modo de comando, el INSERT mostrado en la parte inferior de la pantalla desaparece. Usando los comandos para guardar archivos disponibles en ex, se pasara a modo ex presionando " : " (dos puntos). Aparece " : " al final de la pantalla de vi (donde estaba el INSERT). Luego, se ingresara el comando w para guardar el trabajo. El archivo será guardado en el archivo poem.txt en el medio de almacenamiento secundario.

Nota: en caso del comando :q!, si se olvida el !, vi no lo dejara salir sin guardar su trabajo

6. Eliminar Texto

En esta sección, se aprenderá como eliminar texto en el editor vi y cuales son los comandos que permiten eliminar una palabra o una linea completa.

Texto Ejemplo:

Hi mickeymouse

La Tabla 6.3 ilustra los comandos de eliminación y sus propósitos

Comando	propósito	Ejemplo
x	Elimina un carácter, donde el cursor esta ubicado	Si el cursor esta en 'k', entonces elimina 'k' Resultado: Hi miceymouse
dw	Elimina desde la posición actual del cursor hasta el final de la palabra	Si el cursor esta en 'y', entonces elimina 'ymouse' Resultado: Hi micke
dd	Elimina la linea actual	Si el cursor esta en la linea 'Hi mickeymouse' entonces elimina la linea
D	Elimina desde la posición actual del cursor hasta el final de la linea	Si el cursor esta en 'i' de 'Hi', entonces elimina 'i mickeymouse'

Tabla 6.3: Comandos de Eliminación de Texto de vi

Regresando al poema de ejemplo en la linea cuatro, luego de la palabra 'age-old', hay un '.' (punto). Esto es incorrecto y se quiere eliminar. Se vera como se puede hacer esto.

Simplemente se mueve el cursor hasta el carácter punto y se presiona 'x'. El carácter será eliminado.

Asuma que ahora se quiere eliminar la palabra 'sackles' en la segunda linea del poema. Para hacer esto, se llevara el cursor hasta la primera letra de dicha palabra (la letra s) y

luego se ingresara el comando dw. La palabra será eliminada. El texto se vera como se muestra a continuación. El texto mostrado es luego de eliminar el punto (.) y ‘sackles’.

I wanna break free
From the of this society,
From the rules and regulations governing it,
From its prevailing age-old prototype.

Se presentara otro ejemplo. Colóquese el cursor bajo la letra ‘L’ en la palabra ‘rules’ (tercera linea). Ahora si ingresa el comando dw, elimina las letras a partir de ‘L’ hasta el final de la palabra, pero ‘ru’ permanece, como se muestra a continuación.

I wanna break free
From the of this society,
From the ru and regulations governing it,
From its prevailing age-old prototype.

Si se desea eliminar una linea completa, se puede hacer colocando el cursor al inicio de la linea e ingresando el comando dd. En el ejemplo, si se quiere eliminar la segunda linea, se moverá el cursor hasta el inicio de la linea y se ingresara el comando dd. La linea completa será eliminada. Lo que queda en pantalla se muestra a continuación.

I wanna break free
From the rules and regulations governing it,
From its prevailing age-old prototype.

Vi proporciona una facilidad llamada modificadores de comando para eliminar mas de una letra, palabra o linea usando el mismo comando. Los modificadores de comando mejoran y aumentan el poder de los comandos de eliminación de texto.

La Tabla 6.4 muestra los modificadores de comando y sus usos.

Texto de Ejemplo:

Hello Mickeymouse
Hello Donald
Hello Asterik

Comando	propósito	Ejemplo
nx	Elimina n caracteres a partir de la posición del cursor	Si el cursor esta en ‘c’ de ‘Hello Mickeymouse’ en la primera linea entonces 3x elimina ‘cke’ Resultado: Hello Miymouse Hello Donald Hello Asterik
dnw	Elimina n palabras de la posición actual del cursor	Si el cursor esta en ‘e’ de ‘Hello’ en la primera linea entonces d2w elimina ‘ello’ y ‘Mickeymouse’ Resultado: H Hello Donald Hello Asterik
ndd	Elimina n líneas desde la posición actual del cursor.	Si el cursor esta en ‘H’ de ‘Hello’ en la segunda linea entonces 2dd elimina ‘Hello Donald’ y ‘Hello Asterik’ Resultado: Hello Mickeymouse
DG	Elimina desde la posición actual del cursor hasta el final de dicha linea y lleva el cursor hasta el final del archivo	Si el cursor esta en ‘M’ de ‘Mickeymouse’ en la primera linea entonces elimina ‘Mickeymouse’ y lleva el cursor hasta el final del archivo Resultado: Hello Hello Donald Hello Asterik

DnG	Elimina la linea donde el cursor esta presente y mueve el cursor al inicio de la n-sima linea	Si la posición del cursor es 'H' de 'Hello Donald' entonces D1G elimina 'Hello Donald' y lleva el cursor hasta la 'H' de 'Hello Mickeymouse' Resultado: Hello Mickeymouse Hello Asterik
D$	Elimina a partir de la posición actual hasta el final de la linea	Si el cursor esta en 's' de 'Asterik' entonces d2$ elimina 'ymouse' y 'Hello Donald'. Resultado: Hello Mickeymouse Hello Donald Hello A
dn$	Elimina a partir de la posición actual hasta el final de la linea n especificada	Si el cursor esta en 'y' de 'Hello Mickeymouse' entonces elimina 'sterik'. Resultado: Hello Micke Hello Asterik

Tabla 6.4: Modificadores de Comandos vi.

Para eliminar tres letras de la palabra 'wanna' en la primera linea del poema, se mueve el cursor bajo la letra 'w' y se ingresa el comando 3x. Los tres caracteres desde la posición actual del cursor (letra 'w'), se eliminaran. La pantalla se vera como se muestra a continuación

I na break free
From the of this society,
From the ru and regulations governing it,
From its prevailing age-old prototype.

En el poema, para eliminar la palabra 'ruand regulations', se coloca el cursor al inicio de la palabra (letra 'r' en este caso) y luego se ingresa el comando 2dw. Las dos palabras son eliminadas.

Para eliminar los dos primeras líneas del poema, se coloca el cursor bajo la letra 'I' en la primera linea y se ingresa el comando d2$. La linea será removida y el poema se vera como se muestra a continuación.

From its prevailing age-old prototype.

Una vez estudiados los comandos básicos, se continuara con el aprendizaje del editor vi en detalle. En la siguiente sección, se aprenderá como mover una parte del archivo usando los comandos de movimiento de cursor de vi.

7. Comandos de Movimiento del Cursor

Se iniciara esta sección con movimientos básicos para moverse hacia arriba o hacia abajo o de izquierda a derecha, desde la posición actual del cursor usando los comandos del editor vi.

Nota: también se pueden usar las teclas de dirección para mover el cursor en la pantalla.

7.1 Movimientos del Cursor – Comandos Básicos.

La tabla 6.5 presenta los comandos básicos para el movimiento. Estos se usan para cambiar la posición del cursor en la ventana del editor vi

Texto Ejemplo:

Cricket
Football
Hockey

Comando	propósito	Ejemplo
h	Un espacio a la izquierda de la posición actual del cursor	Si el cursor esta en la 'c' de 'Hockey', lleva al cursor a la 'o' de 'Hockey'

l	Un espacio a la izquierda de la posición actual del cursor	Si el cursor esta en la ‘F’ de ‘Fooball’, lleva al cursor a la primera ‘o’ de ‘Football’
j	Un espacio debajo de la posición actual del cursor	Si el cursor esta en la ‘r’ de ‘Cricket’, lleva al cursor a la primera ‘o’ de ‘Football’
k	Un espacio arriba de la posición actual del cursor	Si el cursor esta en la ‘k’ de ‘Hockey’, lleva al cursor a la ‘t’ de ‘Football’

Tabla 6.5: Movimientos del Cursor - Comandos Básicos

Nota: si se intenta un movimiento imposible, tal como dar el comando k cuando el cursor esta en la primera linea, la pantalla parpadeara o el terminal emitirá un beep. Esto no debe causar alarma dado que el archivo no se dañara.

7.2 Movimientos del Cursor – Comandos de Palabras.

La Tabla 6.6 presenta los comandos de movimiento de cursor por palabras

Texto Ejemplo:

Welcome to Linux

Comando	propósito	Ejemplo
e o E	Mueve el cursor al final de la siguiente palabra, cuando el cursor esta al final de la palabra actual. En caso contrario al final de la misma palabra	Si el cursor esta en la ‘c’ de ‘welcome’, lleva el cursor a ‘e’ al final de ‘welcome’. Si el cursor esta en ‘o’ de ‘to’, lleva el cursor a ‘x’ de ‘Linux’
W o w	Mueve el cursor al inicio de la siguiente palabra	Si el cursor esta en lo ‘o’ de ‘welcome’, lleva el cursor a ‘t’ de ‘to’
B o b	Mueve el cursor al inicio de la palabra anterior, cuando el cursor esta al inicio de la palabra actual. En caso contrario al inicio de la misma palabra	Si el cursor esta en la ‘m’ de ‘welcome’, lleva el cursor a ‘w’ al final de ‘welcome’. Si el cursor esta en ‘t’ de ‘to’, lleva el cursor a ‘w’ de ‘welcome’

Tabla 6.6: Movimientos del Cursor – Comandos de Palabra

7.3 Movimientos del Cursor – Comandos de líneas.

La Tabla 6.7 presenta los comandos de movimiento del cursor por linea.

Texto Ejemplo:

This is test line
This is next test line

Comando	propósito	Ejemplo
0 (cero)	Mueve el cursor al inicio de la linea	Si el cursor esta en la ‘l’ de ‘line’ en la primera linea, lleva el cursor a ‘T’ de ‘This’
^	Mueve el cursor a la primera palabra de la linea	Si el cursor esta en la ‘e’ de ‘test’ en la primera linea, lleva el cursor a ‘T’ de ‘This’
$	Mueve el cursor al final de la linea	Si el cursor esta en la ‘i’ de ‘This’ en la primera linea, lleva el cursor a ‘e’ de ‘line’

<CR>	Mueve el cursor al inicio de la siguiente linea. (CR del ingles 'carriage return'- retorno de carro), que equivale a presionar la tecla <ENTER>	Si el cursor esta en la 'n' de 'line' en la primera linea, lleva el cursor a 'T' de 'This' en la segunda linea.

Tabla 6.7: Movimientos del Cursor – Comandos de Linea

7.4 Movimientos del Cursor – Comandos de Archivos.

La Tabla 6.8 presenta los de movimiento del cursor para comandos de archivos

Texto Ejemplo:

This is test line
This is next test line

Comando	Propósito	Ejemplo
G	Mueve el cursor al primer carácter de la ultima linea	Si el cursor esta en la 'i' de 'is' en la primera linea, lleva el cursor a 'T' de 'This' en la segunda linea.
1G	Mueve el cursor al primer carácter de la primera linea	Si el cursor esta en la 's' de 'test' en la segunda linea, lleva el cursor a 'T' de 'This' en la primera linea.

Tabla 6.8: Movimientos del Cursor – Comandos de Archivos

7.5 Movimientos del Cursor – Comandos de Pantalla.

La pantalla se desplazara automáticamente cuando el cursor alcance ya sea el inicio o el final de una pantalla. Hay algunos comandos adicionales, los cuales permiten controlar el desplazamiento, estos comandos se ilustran en la Tabla 6.9

Comando	propósito
<ctrl f>	Avanza (una pantalla completa)
<ctrl b>	Retrocede (una pantalla completa)
<ctrl d>	Desplaza hacia abajo (media pantalla)
<ctrl u>	Desplaza hacia arriba (media pantalla)

Tabla 6.9: Movimientos del Cursor – Comandos de Pantalla

7.6 Movimientos del Cursor – Otros Comandos.

La Tabla 6.10 presenta otros comandos de movimiento del cursor

Texto Ejemplo:

This is first line
{
This is second line
}
This is third line

Comando	Propósito	Ejemplo
nG	Mueve el cursor al numero de linea n.	Por ejemplo, 3G lleva el cursor a la tercera linea. 'This is second line'
<ctrl G>	Muestra el numero de linea actual	Si el cursor esta 'This is second line', entonces se muestra como resultado el numero de linea 3 of 5 –60% --col 1.
%	Mueve el cursor a la llave, paréntesis o corchete correspondiente	Lleva el cursor de la llave '{'de apertura a su correspondiente llave '}' de cierre. Este comando también sirve para ( ) [ ].

n| (n seguido por el símbolo de tuberías |)	Mueve el cursor a la columna n, donde n es un entero	Si el cursor esta en la ‘s’ de ‘second’ en la tercera linea, entonces 3| lleva al usuario a ‘i’ de ‘This’ en la tercera linea.
n1 (n seguido por 1)	Mueve el cursor n columnas a la derecha del cursor	Si el cursor esta en la ‘s’ de ‘second’ en la tercera linea, entonces 41 lleva al usuario a ‘n’ de ‘second’ en la tercera linea.

Tabla 6.10: Movimientos del Cursor – Otros Comandos.

8. Modificación de Texto

Vi ofrece un conjunto muy grande de comandos para ayudar a cambiar el contenido de un archivo. En esta sección, se discutirán algunos de estos comandos.

8.1 Deshacer Cambios

En ocasiones, se podría necesitar deshacer los cambios realizados. Los siguientes comandos restauran el texto antes de los cambios como se muestra en la Tabla 6.11

Texto Ejemplo:

Atómica

Comando	propósito	Ejemplo
u	Deshace el ultimo comando	Si se agrega ‘key’ a ‘Atómica’, revierte al texto original ‘Atómica’
U	Deshace los cambios de la linea actual	Si se cambia la linea de ‘Atómica’ a ‘Cell’, revierte al texto previo ‘Atómica’
:e!	Editar de nuevo. Restaura el texto al estado que tenia la ultima vez que se grabo	Si se agrega ‘l’ a ‘Atómica’ haciendo ‘Atomical’ y se graba, este comando revertirá al texto anterior ‘Atómica’

Tabla 6.11: Comandos para Deshacer

Vi no solo permite deshacer cambios, sino también rehacer cambios.

8.2. Otros Comandos de Modificación de Texto.

Vi ofrece comandos, que permiten realizar reemplazos en el texto de manera instantánea, ahorrando el problema de primero eliminar y luego escribir la nueva versión. A continuación se discutirá el resto de los comandos de modificación.

Texto Ejemplo:

This Server
This Client

Comando	propósito	Ejemplo
r	Reemplaza el carácter donde esta situado el cursor, con una letra	Si el cursor esta en ‘S’ de ‘Server’ entonces ingresar rC reemplaza ‘S’ con ‘C’ para ser ‘Cerver’ Resultado: This Cerver This Client
R	Reemplaza el texto con un nuevo texto	Si el cursor esta en ‘v’ de ‘Server’ entonces ingresar R key cambia el texto ‘key’ por ‘ver’ para ser ‘Serkey’ Resultado: This Serkey This Client
cw	Cambia la palabra actual	Si el cursor esta en ‘Server’ entonces ingresar cw key para ser ‘This key’ Resultado: This key This Client

c$	Cambia el texto de la posición actual hasta el final de la linea	Ayuda a eliminar letras desde 'r' en 'Server' hasta el final de la linea y cambiarlas por el texto 'key'. Esto hace la nueva palabra 'This Sekey' Resultado: This Sekey This Client
cnw ncw	Cambia las siguientes n palabras. (Igual que cw)	Por ejemplo, c2w ayuda a cambiar el texto 'This Server por otro texto diferente, tal como 'That Cell' Resultado: That Cell This Client
cn$ nc$	Cambia hasta el final de la linea n	Si el cursor esta en 'r' de 'Server', entonces c2$ ayuda a cambiar el texto 'rver' por otro texto diferente, como 'key' y además la siguiente linea 'This Client' con un texto, tal como 'This Router' Resultado: This Sekey This Router
C	Cambia hasta el final de la linea	Si el cursor esta en 'e' de 'Client' y el nuevo texto es 'key', entonces se vuelve 'Clikey' Resultado: This Server This Clikey
cc	Cambia la linea actual	Cambia toda la linea, por ejemplo, de 'This Server' a 'That Server Resultado: That Server This Client
s	Sustituye el carácter actual por el texto ingresado	Cambia la letra 'C' de 'Client' con 'P' para hacer 'Plient' Resultado: This Server This Plient
ns	Sustituye el texto ingresado en los n siguientes caracteres	Ingresando 3s en el texto 'Server' con el cursor en 'e' y el nuevo texto 'key', lo hace 'Skeyer' Resultado: This Skeyer This Client
S	Reemplaza toda la linea	Si el cursor esta bajo 'C' de 'Client' y si se ingresa S seguido por Server, cambiara la linea a 'Server'. Al presionar S, se elimina toda la linea

Tabla 6.12: Comandos de Modificación de Texto

En la siguiente sección se discutirá como copiar una parte del texto desde un archivo y reutilizarlo en otras partes del archivo.

8.3. Copiar y Pegar Texto.

Copiar texto involucra los tres siguientes pasos principales:

- Copiar el texto a un buffer. Un buffer es una ubicación de almacenamiento temporal para guardar texto
- Mover el cursor al lugar de destino.
- Pegar (colocar) el texto del buffer de edición

La Tabla 6.13 presenta los comandos de copia

Texto Ejemplo:

I am Happy
I am Wise

Comando	propósito	Ejemplo
yy	Mueve una copia de la linea actual al buffer sin nombre	Si el cursor esta en 'I' de 'I am Happy', entonces copia 'I am Happy' a un buffer sin nombre

Y	Mueve una copia de la linea actual al buffer sin nombre	Si el cursor esta en ‘I am Happy’, entonces copia ‘I am Happy’ a un buffer sin nombre
nyy	Mueve las n líneas siguientes al buffer sin nombre	Si el cursor esta en ‘I am Happy’, entonces 2yy copia ‘I am Happy’ y ‘I am Wise’ a un buffer sin nombre
nY	Mueve las n siguientes líneas al buffer sin nombre	Si el cursor esta en ‘I am Happy’, entonces 2Y copia ‘I am Happy’ y ‘I am Wise’ a un buffer sin nombre
yw	Mueve una palabra al buffer sin nombre	Si el cursor esta en ‘H’ de ‘Happy’ entonces copia ‘Happy’ a un buffer sin nombre.
nyw	Mueve n palabras al buffer sin nombre	Si el cursor esta en ‘a’ de ‘am’ en la primera linea, entonces y2w copia ‘am’ y ‘Happy’ a un buffer sin nombre
ynw	Mueve n palabras al buffer sin nombre	Si el cursor esta en ‘a’ de ‘am’ en la primera linea, entonces 2yw copia ‘am’ y ‘Happy’ a un buffer sin nombre
y$	Mueve la posición del cursor al final de la linea	Si el cursor esta en ‘I’ de ‘I am Happy’ se mueve a ‘y’, que esta al final de esa linea

Tabla 6.13: Comandos de Copia

La Tabla 6.14 presenta los comandos de Pegado

Texto Ejemplo:

I am Happy
I am Wise

Comando	propósito	Ejemplo
P	Pega del buffer sin nombre a la derecha del cursor	Pega, por ejemplo, ‘Happy’ almacenado en el buffer sin nombre, al lado derecho del cursor
p	Pega del buffer sin nombre a la izquierda del cursor	Pega, por ejemplo, ‘Happy’ almacenado en el buffer sin nombre, al lado izquierdo del cursor
nP np	Pega n copias del buffer sin nombre a la izquierda del cursor	2p pega, ‘Happy’ almacenado en el buffer sin nombre, dos veces al lado izquierdo del cursor

Tabla 6.14: Comandos de Pegado

Dado que el buffer sin nombre se puede corromper fácilmente por el uso de otros comandos comunes, se pueden usar los buffers nombrados. Vi tiene 26 buffers con nombre. Usan las letras del alfabeto como identificación

El carácter ‘u’ se usa para distinguir los buffers con y sin nombre.

La Tabla 6.15 muestra los comandos de copia y pegado de buffers con nombre.

Texto Ejemplo:

This is Cat
This is Dog

El <char> usado en la siguiente tabla es cualquier carácter en minúscula que no es un comando vi tal como i, a, e, u, etc.

Comando	propósito	Ejemplo
“<char>yy	Mueve la linea actual al buffer con nombre <char>	Si el cursor esta en ‘T’ de ‘This’ en la primera linea, copia ‘This is Cat’ al buffer con nombre, por ejemplo, m

“<char>Y	Mueve la linea actual al buffer con nombre <char>	Si el cursor esta en ‘T’ de ‘This’ en la primera linea, copia ‘This is Cat’ al buffer con nombre, por ejemplo, m
“<char>yw	Mueve la palabra actual al buffer con nombre <char>	Si el cursor esta en ‘C’ de ‘Cat’ en la primera linea, copia ‘Cat’ al buffer con nombre, por ejemplo, m
“<char>yw (<char> en mayúsculas)	Agrega la palabra al contenido del buffer con nombre <char>	Si el cursor esta en ‘D’ de ‘Dog’ en la primera linea, agrega ‘Dog’ al buffer con nombre, por ejemplo, m, que ya contenía ‘Cat’
“<char>y2w	Mueve las dos siguientes palabras al buffer <char>	Si el cursor esta en ‘T’ de ‘This’ en la primera linea, entonces my2w ‘This is’ al buffer con nombre, por ejemplo, m
“<char>p	Pega desde el buffer con nombre char a la derecha del cursor	Pega el buffer con nombre, por ejemplo, m, que contiene por ejemplo ‘Cat’ al lado derecho del cursor
“<char>nP	Pega n copias del buffer con nombre <char> a la izquierda del cursor	M3P pega tres copias del buffer con nombre m, que contiene por ejemplo, ‘Dog’ al lado izquierdo del cursor

Tabla 6.15: Comandos de Copia de Buffer con Nombre

8.4. Copiar y Pegar Texto.

Cortar y pegar texto consiste de los tres siguientes pasos:

- Eliminar el texto a un buffer con o sin nombre
- Mover el cursor a la ubicación destino
- Pegar el buffer con nombre o sin nombre

El proceso es el mismo que copiarlo, pero con un cambio en el primer paso, de copiar a eliminar, que se presenta en la Tabla 6.16.

Texto Ejemplo

My Book
My Pen

Comando	propósito	Ejemplo
“bdd	Elimina la linea y la coloca en el buffer con nombre b	Si el cursor esta en ‘M’ de ‘My Book’ elimina dicha linea y la copia al buffer con nombre b.
“B2dd	Elimina dos líneas y la coloca en el buffer con nombre B	Si el cursor esta en ‘M’ de ‘My Book’, elimina ‘My Book’ y ‘My Pen’ y copia ambos al buffer con nombre B.
“dw	Elimina una palabra y la coloca en el buffer sin nombre	Si el cursor esta en ‘P’ de ‘My Pen’ elimina la palabra ‘Pen’ y la copia al buffer sin nombre.

Tabla 6.16: Comandos para Cortar y Pegar

En el caso de que el sistema colapse, el contenido de los buffer con y sin nombre se pierde, pero el contenido del buffer de edición puede ser recuperado.

9. Buscar Texto

Vi tiene cierto numero de comandos de búsqueda. La Tabla 6.17 muestra los comandos de búsqueda que se usan comúnmente.

Texto Ejemplo:

The Client Machine
The Server Machine

Comando	propósito	Ejemplo
fc	Encuentra el siguiente carácter c a la derecha del cursor en la misma linea	Si el cursor esta en ‘C’ de ‘Client’ y se usa fa, el cursor se mueve hasta ‘a’ de ‘Machine’ de la primera linea
Fc	Encuentra el siguiente carácter c a la izquierda del cursor en la misma linea	Si el cursor esta en ‘c’ de ‘Machine’ en la segunda linea y se usa FT, el cursor se mueve hasta ‘T’ de ‘The’ de la segunda linea
tc	Encuentra el carácter antes del siguiente carácter c a la derecha del cursor en la misma linea	Si el cursor esta en ‘C’ y se usa ta, el cursor se mueve hasta ‘M’ de ‘Machine’ de la primera linea
Tc	Encuentra el carácter después del siguiente carácter c a la derecha del cursor en la misma linea	Si el cursor esta bajo ‘c’ de ‘Machine’ en la segunda linea y se usa TS, el cursor se mueve hasta ‘e’ luego de ‘S’ de ‘Server’ de la segunda linea
; (punto y coma)	Repite el ultimo f, F, t, T hacia delante	Si el cursor esta en ‘C’, y se usa fe, el cursor se mueve al primer ‘e’ a la derecha. Al presionar ; se mueve al siguiente e, si lo encuentra, en la misma linea hacia la derecha
, (coma)	Repite el ultimo f, F, t, T hacia atrás	Igual que ; (punto y coma), pero invierte la dirección del comando original

Tabla 6.17: Comandos de búsqueda

Nota: Los comandos que empiezan con f, F, t, T son aplicables solo para la linea actual. Si el carácter que se esta buscando no se encuentra, vi emitirá un sonido o dará alguna otra clase de señal.

Vi le permite buscar una cadena en el buffer de edición. La Tabla 6.18 ilustra los comandos relacionados a esto.

Texto Ejemplo:

This Client machine is on the desk of the Client
This Client is happy with the machine

Comando	propósito	Ejemplo
/cadena	Encuentra la siguiente ocurrencia de cadena	Si el cursor esta en ‘C’ de la primera palabra ‘Client’ en la primera linea, ingresando /Client se moverá el cursor hacia ‘C’ de ‘Client’, que es la ultima palabra en la linea actual (la siguiente ocurrencia de cadena) en el archivo
?cadena	Encuentra la ultima ocurrencia de cadena	Si el cursor esta en ‘C’ de la primera palabra ‘Client’ en la primera linea, ingresando ?Client se moverá el cursor hacia ‘C’ de ‘Client’, que es la segunda palabra en la segunda linea (la siguiente ocurrencia de cadena) en el archivo
n	Repite el ultimo comando / o ? hacia adelante	Para la /cadena usada antes, n repite y encuentra el siguiente Client, que es la segunda palabra en la segunda linea.

N	Repite el ultimo comando / o ? hacia atrás	Igual que n, pero invierte la dirección del comando original

Tabla 6.18: Comandos de búsqueda de Cadena

Cuando se usa los comandos / o ?, se limpiara una linea al final de la pantalla.
La cadena en el comando / o ? puede ser una expresión regular. Una expresión regular es una descripción de un conjunto de caracteres. La descripción se construye usando texto mezclado con caracteres especiales. Los caracteres especiales en las expresiones regulares son. * [ ] ^ $

Unidad 7: Laboratorio del Editor vi

Objetivos de Aprendizaje

Al final de esta unida Ud. será capaz de:

- Usar el editor vi.
- Emplear diferentes grupos de comandos en vi
- Editar texto simple

Ejercicios de Laboratorio

1) Cree un archivo de texto llamado vi.txt e ingrese el siguiente texto en el.

Los editores se usan para crear documentos. Linux ofrece varios editores; algunos de ellos son vi, emacs, ed, ex. Los editores como ed y ex se llaman editores de linea. Mientras que ed permite realizar inserción, eliminación, modificación, etc, linea por linea; vi ofrece una facilidad de pantalla completa para crear y editar documentos. Se ha discutido en detalle acerca de ed y sus capacidades. En esta unidad, se discutirá el editor vi.

El termino vi viene de 'visual editor'. Vi (se pronuncia 'vee eye') es el único editor que se encuentra en casi cada instalación de Unix/Linux. Escrito originalmente en la Universidad de California en Berkeley, debe su origen a William Joy. Sus versiones pueden encontrarse ahora en casi cada edición de venta de Linux. Vi empezó como el editor de linea ex y creció a lo largo de los años hasta convertirse en un poderoso editor visual. Ex ahora existe en forma separada y es un modo de edición especial de vi. Los comandos de ex pueden ser iniciados a traves de vi.

Vi proporciona un amplio conjunto de comandos para insertar, eliminar y modificar texto. Usando expresiones regulares se pueden realizar poderosas búsquedas y reemplazos de texto. Las expresiones regulares son formulas que hacen coincidir cadenas con un patrón dado.

Vi proporciona tres métodos en que un usuario puede operar. Ahora se aprenderán los modos en vi.

2) Practique los comandos que aprendió en la unidad anterior. Siga la misma secuencia dada en la unidad anterior, de manera que todos los comandos se usen. Al final de esta unidad, Ud debe ser capaz de crear, modificar y visualizar un archivo de texto simple usando vi.

Volumen 2
Shell y Procesos en Linux

Unidad 1: Fundamentos de Shell

Objetivos de Aprendizaje

Al final de esta Unidad, Ud será capaz de:

- Proporcionar una visión general del shell de Linux
- Discutir acerca de las variables shell
- Describir los metacaracteres
- Explicar como asignar y referenciar las variables shell

1. Introducción al Shell de Linux

En el volumen 1 Unidad 1 – *Fundamentos de Linux*, se discutió brevemente acerca del shell (interprete de comandos). En esta unidad se estudiaran varios aspectos del shell de Linux, las variables proporcionadas por el shell y los diferentes metacaracteres utilizados por el mismo.

El shell es una de las herramientas más poderosas de los sistemas Linux. Como se explico anteriormente, el shell es un programa escrito en C que ejecuta un conjunto de comandos Linux y procesa las entradas del usuario. Empieza cuando el usuario ingresa (login) y termina cuando el usuario se desconecta (logout). Proporciona una interfaz para el sistema operativo subyacente.

El shell interpreta los comandos ingresados por los usuarios y los comunica al núcleo (kernel). El kernel maneja las interacciones complejas con el hardware del sistema. Algunos ejemplos de interacciones complejas son abrir y cerrar archivos, imprimir, etc. Estas interacciones son complejas porque necesitan trabajar con los recursos soportados por el sistema operativo a bajo nivel. El usuario del sistema operativo Linux funciona a alto nivel, de modo que al ingresar el comando en el indicador (prompt), se produce el resultado deseado. Para imprimir un documento, el usuario tiene que invocar un comando que maneja el trabajo de impresión. El kernel transmite los bytes a traves del cable de la impresora a la impresora para imprimir y el usuario no necesita preocuparse del manejo a bajo nivel sobre como imprimir un documento. El shell actúa como un intermediario entre el usuario y el kernel.

Algunas de las responsabilidades del shell son:

- Controlar y asignar trabajos
- Manejar mas de un proceso a la vez
- Redireccionar entrada y salida estándar
- Mantener un historial de comandos
- Proporcionar un lenguaje de comandos para escribir *shell scripts*

Los shell scripts son programas escritos usando sintaxis del lenguaje de comandos del shell. De hecho, es solo otro lenguaje de programación capaz de ejecutar comandos a nivel del shell.

Los shell scripts permiten al usuario “poner lotes” diversos comandos del shell en un archivo. Hay un número de shells disponibles en Linux como el shell BASH, el shell C, etc., como se menciono en las unidades anteriores.

La diferencia más importante entre los shells, es el lenguaje de comandos que ellos proporcionan. El shell C proporciona sintaxis muy similar al lenguaje de programación C. el BASH (Bourne Again Shell) utiliza una sintaxis diferente.

En esta unidad, se explicara el Linux BASH y las características que lo hacen interesante. Se comenzara la discusión con las variables shell.

2. Variables Shell

El shell mantiene un número de variables, llamadas *variables shell*, para mantener el registro de diferentes tipos de información. Las variables shell son locales al shell en el que estén definidas.

El shell utiliza variables para mantener el registro del usuario y la información del sistema. Estas variables son especificas al shell en que el usuario esta trabajando. Cada shell tiene su propio conjunto de variables, que pueden ser asignadas y referenciadas por el usuario. También hay variables de entorno que son visibles a traves de todos los shells.

Una de las variables shell es PATH, la cual se explicara brevemente para entender otras variables shell.

La mayoría de los comandos que se ejecutan desde el directorio home, están ubicados en otro lugar. Para ubicar estos comandos, la variable PATH en Linux se establece en todos los directorios del sistema donde los comandos estén disponibles para su ejecución. Así,

cuando se ingresa cat en el prompt, la información con respecto a la ubicación del comando cat se obtiene de la variable PATH y luego se ejecuta. La información disponible en PATH consiste simplemente en la ruta completa de los directorios donde los comandos están ubicados.
Se puede ver el contenido de todas las variables shell ingresando un símbolo $ (de peso), antes de la variable. De esta manera

```
edwar@localhost:~> echo $PATH
        /opt/java/bin/:/opt/java/bin:/home/edwar/bin:/usr/local/bin:/usr/X11R6/bin:/usr/games
        :/opt/kde/bin:/gnome/bin:/usr/lib/mit/sbin:/usr/lib/qt3/bin:/opt/IBMJava2-
        142/bin:/usr/local/lib/xerces-c-src_2_7_0/bin:/usr/local/pgsql/bin
edwar@localhost:~>.
```

Lo anterior es el contenido de la variable PATH en un SuSE Linux 10.3 que se uso para probar los ejemplos expuestos en este texto. El contenido podría ser diferente en otros sistemas. Cada ruta de directorio esta separada por dos (:) puntos.
PATH es solo una de muchas variables predefinidas que mantiene un shell. En la Unidad 3: *Características de Shell* se estudiaran otras variables predefinidas.

2.1 Variables Shell Definidas por el Usuario

El shell permite a los usuarios crear sus propias variables. Estas se denominan variables shell definidas por el usuario. El termino “definidas por el usuario” es para distinguir estas variables de las variables predefinidas por el shell. Para crear una variable, el usuario debe ingresar:

```
        name=value
```

Donde name es la variable shell definida por el usuario y value es el valor asignado a la variable shell definida por el usuario. Note que no debe existir espacio antes ni después del signo de igualdad (=).

```
edwar@localhost:~> color=azul
edwar@localhost:~>.
```

La variable color ahora es definida por el usuario y le es asignado el valor azul.
Para ver el valor de la variable shell definida por el usuario, se usa el signo $ junto con el nombre de la variable. Usando el comando echo, se puede imprimir el valor establecido por el usuario.

```
edwar@localhost:~> echo $color
azul
edwar@localhost:~>
```

A continuación más ejemplos:

```
edwar@localhost:~> m00=100
edwar@localhost:~> echo $m00
100
edwar@localhost:~>
edwar@localhost:~> 100=100
bash: 100=100 command not found (archivo no encontrado)
edwar@localhost:~>.
```

Primero se establece la variable definida por el usuario m00 a 100. Cuando se intento establecer la variable definida por el usuario 100 con un valor 100, el shell presento un mensaje de error. Esto es por que los nombres de las variables definidas por el usuario no pueden comenzar con un número. A continuación se presentan los diferentes tipos de metacaracteres proporcionados por el sistema operativo Linux.

3. Metacaracteres

Los metacaracteres son caracteres que tienen un significado especial para el shell. Algunos de los metacaracteres disponibles son:

```
<>
|
```

;
?
*
[ ]
$
\
( )
{ }
“
‘

Se encontraron algunos caracteres, >, <, >> y |, en el Volumen 1 Unidad 2 – *El Sistema Linux*. En dicha unidad se les refería a ellos como operadores de redirección. Para facilitar su comprensión, los metacaracteres pueden agruparse en las siguientes categorías:

- Sustitución de Comodines
- Redirección y Tuberías (pipes)
- Notación de linea de comandos
- Sustitución de Parámetros
- Escapes y comillas

Una lista completa de metacaracteres se muestra al final de esta unidad.

3.1 Sustitución de Comodines

Los caracteres de comodín son un subconjunto de metacaracteres que se usan para buscar e igualar a un patrón de archivo. Los cuatro caracteres comodín que se aprenderán aquí, son:

- ?: este metacaracter equivale a cualquier carácter en un nombre de archivo
- *: este metacaracter equivale a cualquier cadena de cero o mas caracteres en un nombre de archivo
- [list]: es equivalente a cualquier carácter list. Aquí list puede contener uno mas caracteres como [asdf]
- [^list]: es equivalente a cualquier carácter que no este en list

Para entender estos metacaracteres, se asumirá que el directorio de inicio (home) de edwar tiene los siguientes archivos y directorios

linuxChap1.doc
linuxChap2.doc
linuxCont.doc
linux1.txt
linux2.txt
linuxInternals
unixInternals
link.txt
listing.lst
mist.doc
nest.lst

linuxInternals y unixInternals son subdirectorios de edwar.

En esta unidad, se proporcionaran los ejemplos de muchas variables shell. Estos ejemplos aparecerán entre ‘Uso de shvar’ y ‘Fin de uso de shvar’, donde shvar será la variable shell que se este estudiando.

Uso de ?

edwar@localhost:~> ls –o
total 40
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 lint.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux1.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux2.txt

-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap1.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap2.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxCont.doc
drwxr-xr-x 2 edwar 2 2009-04-03 09:18 linuxInternals
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 listing.lst
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 mist.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 nest.lst
drwxr-xr-x 2 edwar 48 2009-04-03 09:18 unixInternals
edwar@localhost:~>
ls –o muestra un listado largo de archivos sin información de grupo.
edwar@localhost:~> ls –o linux?.txt
-rw-r- -r- - 1 edwar 2 2009-04-03 09:18 linux1.txt
edwar@localhost:~>

En el ejemplo anterior solo se muestra una linea porque el metacaracter ? equivale exactamente a un carácter en la posición donde apareció el metacaracter ? en el comando.

edwar@localhost:~> ls –o ????.txt
-rw-r- -r- - 1 edwar 2 2009-04-03 09:18 linux1.txt
edwar@localhost:~>

El nombre del archivo linux.txt tiene cinco caracteres antes del punto (.) y así corresponde a los cinco metacaracteres ? especificados con ls –o. No se muestran otros archivos, ya que ninguno de lo otros tiene el patrón especificado en el comando. El modelo es cinco metacaracteres ?, seguidos por un punto (.) y luego txt.

edwar@localhost:~> ls ?
/bin/ls: ?: no existe el fichero o el directorioedwar@localhost:~>

Un nombre de archivo con un solo carácter no existe en el directorio de inicio edwar

Uso de *

edwar@localhost:~> ls –o linux*
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux1.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux2.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap1.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap2.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxCont.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux.txt
linuxInternals:
total 0
edwar@localhost:~>

Muestra todos los archivos y directorios que tiene linux como patrón de inicio. Se observa que Linux muestra el directorio linuxInternals en una linea con total 0 debajo. Dado que se encontró un nombre de directorio con el patrón linux al inicio, se muestran los archivos de ese directorio. En este caso, no se encontraron archivo o directorios en linuxInternals y por eso se observa total 0.

edwar@localhost:~> ls –o linux* .txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux1.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux.txt
edwar@localhost:~>

linux.txt esta incluido en la salida, dado que el metacaracter * implica 0 o mas caracteres. Seguidamente, se muestran algunos ejemplos.

```
edwar@localhost:~> ls –o linuxC* .doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap1.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap2.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxCont.doc
edwar@localhost:~> ls –o * .doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux2.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap1.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap2.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxCont.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 mist.doc
edwar@localhost:~> ls –o *Co*
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxCont.doc
edwar@localhost:~> ls –o linux?.*
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux1.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux2.doc
edwar@localhost:~>
```

Se pueden combinar metacaracteres para obtener cualquier tipo de listado

Uso de [list] y [^list]

```
edwar@localhost:~> ls –o li[ns]t*
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 lint.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 listing.lst
edwar@localhost:~>
```

Este busca n o s en la tercera posición

```
edwar@localhost:~> ls –o *[12C]*
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux1.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux2.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap1.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap2.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxCont.doc
edwar@localhost:~>
```

Este muestra todas las líneas que tienen 0 o mas caracteres seguidos por cualquiera de los caracteres de la lista 12C, seguidos por 0 o mas caracteres nuevamente

```
edwar@localhost:~> ls –o [^lu]*
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 mist.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 nest.lst
edwar@localhost:~>
```

Se esta usando el metacaracter comodín, que busca todos los caracteres de manera que ni l ni u estén en la primera posición.

```
edwar@localhost:~> ls –o [^mnu]inux[12]*
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux1.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux2.doc
edwar@localhost:~>
```

Se puede obtener el resultado anterior con linux [12]*. Pero si se tenía un archivo de nombre sinux.txt o binux.doc habrían sido ignorados.
también se puede usar [a-d], que buscara todos los caracteres en el rango desde a hasta d.

Para entender los conceptos de redirección y tuberías (pipe), se van a introducir primero los conceptos de descriptores de archivos.

3.2 Descriptores de Archivo

Cada programa tiene un descriptor asociado. El shell asigna 3 descriptores estándares cuando el programa empieza. Estos son STDIN, STDOUT, STDERR. Estos tres descriptores se asignan con un valor como se muestra a continuación:

- STDIN 0 StandardInput (Teclado)
- STDOUT 1 StandardOutput (Monitor)
- STDERR 2 StandardErr (Diseccionado al monitor por defecto)

3.3 Redirección y Tuberías

En el Volumen 1, Unidad 2: *El Sistema Linux* se discutió acerca de la redirección y el uso de tuberías (pipes). Se aprendió a usar >, < y >>. Estos se denominan metacaracteres de redirección y | metacaracter de tubería. A continuación dos usos adicionales de redirección, n> >&n.

Para mostrar los usos se asume que el archivo linux.txt contiene las siguientes líneas.

Linux es un sistema operativo
Sigue los estándares GNU.
Linux esta provisto de un lenguaje de comandos
El kernel de Linux es el núcleo del sistema operativo
Linux es un poderoso y versátil sistema operativo.

Uso de n>

n>file
Este comando, 'n' es por descriptor de archivos como se vio en la sección anterior. El nombre de archivo que sigue a > es el archivo destino al que se desea enviar el resultado.
Por defecto, los mensajes de error son redireccionados a la salida, es decir, al monitor. Si se desea redireccionar el error a un archivo es posible usar el operador anterior como se muestra a continuación:

edwar@localhost:~> cat linux.txt archivo.txt
Linux es un sistema operativo
Sigue los estándares GNU.
Linux esta provisto de un lenguaje de comandos
El kernel de Linux es el núcleo del sistema operativo
Linux es un poderoso y versátil sistema operativo
cat: archivo.txt: No existe el fichero o directorio
edwar@localhost:~>

Como no existe archivo.txt, después de que las líneas de linux.txt se muestra un mensaje de error apropiado.
Si se desea capturar el mensaje de error en otro archivo, se puede hacer lo siguiente:
edwar@localhost:~> cat linux.txt archivo.txt 2>error
Linux es un sistema operativo
Sigue los estándares GNU.
Linux esta provisto de un lenguaje de comandos
El kernel de Linux es el núcleo del sistema operativo
Linux es un poderoso y versátil sistema operativo
edwar@localhost:~>

No se ve ahora el mensaje de error n> file se reemplaza por 2>error, donde 2 es el numero de descriptor de archivo para el error estándar mencionado en la sección anterior, el cual se ha redireccionado al archivo llamado error.

edwar@localhost:~> cat error
cat: archivo.txt: No existe el fichero o el directorio.
edwar@localhost:~>

Este tipo de redirección permite a los usuarios grabar mensajes de error cuando los programas están corriendo en segundo plano (background). Más adelante se discutirá como correr programas en segundo plano.

Uso de >&n

En el caso de requerir redireccionar tanto la salida como el error estándar hacia el mismo archivo, se utiliza >&n. Esto es útil cuando se ejecutan programas en segundo plano, donde la salida y el error son capturados en el mismo archivo.
El shell proporciona el mecanismo para combinar estos dos flujos.

```
edwar@localhost:~> cat linux.txt archivo.txt > outfile 2>&1
edwar@localhost:~>
```

No se ve ninguna salida dirigida a la pantalla. Tanto la salida y el error estándar se han redireccionado al archivo outfile.
Aquí la sintaxis es muy importante. Primero se redirecciona la salida estándar a outfile. 2>&1 le dice al shell que el error estándar sea el mismo que la salida estándar. Como se redirecciona la salida a outfile, este se convierte en la salida estándar en este punto
Observe el siguiente comando:

```
edwar@localhost:~> cat linux.txt archivo.txt 2>&1 > outfile
cat: archivo.txt: No existe el fichero o directorio
edwar@localhost:~>
```

Se observa que el error se muestra en el monitor, mientras la salida se envía a outfile
El dispositivo de error se hace corresponder primero al monitor, como el descriptor de archivo 1 es para el monitor en este punto. La salida se redirecciona al archivo outfile después de que se hace corresponder el error estándar al monitor. Así el mensaje de error en la pantalla y la salida se encuentran en el archivo outfile.
Esto es equivalente a decir:

```
edwar@localhost:~>  cat linux.txt archivo.txt > outfile
```

Esto es muy útil cuando se desea capturar los mensajes de depuración en un archivo cuando se compilan programas C o C++

3.4 Notación de Linea de Comandos

Se pueden usar un número de metacaracteres en una linea de comandos para realizar algunas tareas especiales en el shell, a saber:

- Los comandos se pueden ingresar en múltiples líneas usando \.
- Los comandos se pueden ejecutar en un segundo plano usando &.
- Se pueden ingresar mas de un comando en una sola linea usando ; (punto y coma) como separador
- Los comandos pueden agruparse usando ( ).
- Se pueden aplicar operaciones condicionales en los comandos, usando && y ||

Estos se explican a continuación con ejemplos.

**Uso de **

El metacaracter \, seguido por la tecla <ENTER>, permite al usuario ingresar parámetros del comando en múltiples líneas. Cuando se presiona \ y <ENTER>, un símbolo > aparece en la siguiente linea para indicar que el comando no esta completo. Continuar con el comando en la siguiente linea es útil cuando los argumentos del comando son largos.

```
edwar@localhost:~> cat >
> miarchivo.txt
This is a new file
ctrl-d
edwar@localhost:~>
```

El metacaracter \ le indica al shell que espere por una entrada adicional. Si se omite el metacaracter, se genera un mensaje de error. también note que el símbolo > aparece donde se ha ingresado miarchivo.txt. Considere el siguiente comando:
cat >\<ENTER> > miarchivo.txt
Esto es equivalente a cat > miarchivo.txt <ENTER> indica que se debe presionar la tecla ENTER del teclado.

```
edwar@localhost:~>   cp \
> linux.txt \
> miarchivo.txt
edwar@localhost:~>
```

En el ejemplo anterior se ha utilizado el metacaracter \ dos veces, y cada vez el símbolo > aparece como se puede observar.

Uso de &

El metacaracter & se usa para ejecutar el comando anterior en segundo plano

```
edwar@localhost:~>   monitor & linux.txt
[1] 2259
        Linux es un sistema operativo
        Sigue los estándares GNU.
        Linux esta provisto de un lenguaje de comandos
        El kernel de Linux es el núcleo del sistema operativo
        Linux es un poderoso y versátil sistema operativo
Numero de usuarios conectados al sistema: 204
[1] Done monitor
edwar@localhost:~>
```

monitor es un programa ejecutable C. Controla el sistema durante cinco segundos y muestra el numero de usuarios del sistema al final de los cinco segundos. Se puede ejecutar este comando en segundo plano ingresando & después del comando.
Ahora monitor se esta ejecutando en segundo plano y el shell le entrega el numero de identidad 2259. A este número se le denomina *Process ID* o *PID*.
Se aprenderá acerca de los procesos en la *Unidad 5 – Procesos en Linux* de este volumen

El otro comando cat linux.txt no se ejecuta en segundo plano. Aquí la salida que se muestra es inmediata. Cuando el programa en segundo plano termina, se muestra la salida del programa y el mensaje Done.
A veces los programas en segundo plano se ejecutan por un largo periodo de tiempo. En este caso, el shell interrumpe el trabajo que se este ejecutando en ese momento, muestra algún mensaje enviado por el programa de segundo plano y muestra Done si se completa exitosamente.

Uso de ;

El metacaracter ; se usa para separar comandos cuando se ingresa mas de un comando en la misma linea, como se muestra a continuación:

```
edwar@localhost:~>  ls –o; cat linux.txt
total 60
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 error
 -rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 lint.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux1.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux2.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap1.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxChap2.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linuxCont.doc
drwxr-xr-x 2 edwar 2 2009-04-03 09:18 linuxInternals
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 linux.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 listing.lst
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 miarchivo.txt
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 mist.doc
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 nest.lst
-rw-r- -r- -r 1 edwar 2 2009-04-03 09:18 outfile
```

drwxr-xr-x 2 edwar 48 2009-04-03 09:18 unixInternals

Linux es un sistema operativo
Sigue los estándares GNU.
Linux esta provisto de un lenguaje de comandos
El kernel de Linux es el núcleo del sistema operativo
Linux es un poderoso y versátil sistema operativo

edwar@localhost:~>

Nota: generalmente, el carácter de nueva linea actúa como un finalizador de comando. Alternativamente ; y & también pueden usarse como finalizadores de comandos.

Uso de ( )

El metacaracter ( ) permite agrupar comandos en la linea de comandos

edwar@localhost:~> (cat linux.txt; date) > datefile
edwar@localhost:~>

datefile contendrá la salida de linux.txt y la fecha del sistema cuando se emite el comando. Asuma que se ha ingresado el comando como se muestra a continuación sin ().

edwar@localhost:~> cat linuc.txt; date > datefile

Entonces, solo la salida date se habrá almacenado en datefile. Agrupar comandos ayuda a combinar la salida de más de un comando en un archivo de salida.

Uso de && y | |

Los usos de && y | | se muestran con algunos ejemplos para completar el tema de notación de linea de comandos. Se requieren de conocimientos de programación para usar && y | | efectivamente. Estos dos metacaracteres son llamados *operadores condicionales*.

&& se usa cuando el segundo comando va a ser ejecutado solo si el primero se cumple
| | se utiliza cuando el segundo comando va a ser ejecutado solo si el primero falla
Cada comando devuelve al shell al estado de la salida. Se devuelve 0 si l comando se ejecuta sin ningún problema (éxito) y distinto de cero si algo va mal. Este estado de la salida le dice al shell si el comando se ha ejecutado con éxito o no. En base al estado de salida, el shell continúa o detiene la ejecución del segundo comando.
Observe la sentencia que sigue:

edwar@localhost:~> cc prog.c && a.out
edwar@localhost:~>

Esto le dice al shell de ejecutar a.out solo si la compilación de prog.cc se ejecuta con éxito, en otro caso no, cc es un compilador de Programas C

edwar@localhost:~> grep unix linux.txt && echo No encontrado
edwar@localhost:~>

Este comando no muestra nada. El patrón unix no se encuentra en linux.txt. Así grep devuelve un valor distinto de 0 al shell, lo cual significa que la ejecución no fue exitosa. Aquí, el comando siguiente && no se ejecuta echo es un comando en Linux que solo muestra en la salida estándar las palabras ingresadas después de echo.

edwar@localhost:~> grep unix linux.txt | | echo No encontrado
No encontrado
edwar@localhost:~>

Ahora se obtiene la salida deseada, ya que el comando | | se ejecuta solo si el primer comando falla.

A continuación se discutirá la sustitución de parámetros.

3.5 Sustitución de Parámetros

El shell proporciona un metodo para definir y sustituir variables usando metacaracteres de sustitución de parámetros { }. Las siguientes son cuatro formas en las que se puede hacer esto:

${variable-word}	Muestra el valor de variable si este existe; en caso contrario muestra el valor word
${variable=word}	Muestra el valor de la variable si este existe; en caso contrario muestra el valor de word y asigna a variable el valor de word
${variable+word}	Si variable esta asignada, muestra word, en caso contrario no muestra nada
${variable?mesg}	Si variable ya esta asignada, muestra el valor, en caso contrario muestra mesg.

A continuación alguno ejemplos para ayudar a entender esto.

Uso de Sustitución de Parámetros

edwar@localhost:~> echo $ {color-verde}
verde
edwar@localhost:~>

Como color no esta definido, se muestra verde

edwar@localhost:~> color=azul
edwar@localhost:~> echo $ {color-naranja}
azul
edwar@localhost:~>

Muestra azul dado que color ya esta definido. En ambos casos la variable color permanece sin cambios

edwar@localhost:~> echo $ {nuevocolor-verde}
verde
edwar@localhost:~>

Muestra verde y le asigna a nuevocolor el valor verde

edwar@localhost:~> echo $ {nuevocolor+amarillo}
amarillo
edwar@localhost:~>
como nuevocolor ya esta asignado, se muestra amarillo, nuevocolor no es modificado
edwar@localhost:~> echo $ {otro+amarillo}
edwar@localhost:~>

Como otro no esta asignado no muestra nada

edwar@localhost:~> echo $ {nuevocolor? no definido}
verde
edwar@localhost:~>

Como la variable color tiene valor asignado, muestra el valor de nuevocolor

edwar@localhost:~> edwar] $ echo $ {otro? no definido}
bash: otro no definido
edwar@localhost:~>

El mensaje no definido es porque la variable otro no esta definida hasta ahora

Al principio de la unidad se explico acerca de las variables shell y las variables definidas por el usuario. Se ha aprendido cuatro formas en las que las variables shell definidas por el usuario se pueden definir y sustituir usando sustitución de parámetros. Antes de aprender acerca del último conjunto de metacaracteres, denominados Escapes y Comillas, se aprenderá como visualizar las variables asociadas con el shell.

Para obtener una lista de todas las variables con las que el shell esta asociada, se puede usar el comando set. El comando set muestra todas las variables actualmente definidas para el shell.

3.6 Escapes y Comillas

El shell le da significados especiales a todos los metacaracteres que reconoce. Por lo tanto, si se desea usar estos caracteres especiales con su significado original, no es posible hacerlo, ya que el intérprete los entiende de forma diferente. Para superar este

problema, se pueden neutralizar los metacaracteres usando escapes y comillas. A continuación se listan tres metacaracteres que permiten neutralizar los metacaracteres:

- **\(Escape)** se usa para desactivar el significado especial de cualquier metacaracter. Por defecto, el shell interpreta los caracteres con el significado especial asociado a este. Sin embargo, en el caso de necesitar usar los metacaracteres como carácter normal, se debe tomar la ruta de escape usando el metacaracter \. El carácter \ debe usarse antes del metacaracter cuyo significado especial debe ser desactivado
- **‘’(Comillas Simples)** desactivan el significado especial de todos los caracteres que se encuentren entre ellas. Generalmente, es útil cuando se usa más de un metacaracter en la misma linea. En lugar de usar \ antes de cada metacaracter en la linea, se puede encerrar la linea entera con “
- **“”(Comillas Dobles)** se usan para obtener la salida de la ejecución del comando encerrado en ellas.

**Uso de **

```
 edwar@localhost:~>  echo Enter * now
Enter lint.txt linux.txt linux1.txt linux2.doc linuxChap1.doc linuxCahp2.doc linuxCont.doc linuxInternals listing.lst mist.doc nest.lst unixInternals now
edwar@localhost:~>
```

Esto no es lo que se deseaba. Se quería el comando echo para simplemente mostrar Enter * now
El shell entiende o interpreta el * como mostrar todos los archivos, que es precisamente lo que ha hecho. Ingrese las palabras Enter y now, se obtienen los nombres de todos los archivos del directorio actual. Se puede rectificar esto usando el metacaracter \, el cual informa al shell que debe dar el significado original a los caracteres especiales.

```
 edwar@localhost:~>   echo Enter \* now
Enter * now
edwar@localhost:~>
```

Se ha neutralizado el metacaracter * ingresando \ inmediatamente antes de *. De manera similar, se puede usar el \ para neutralizar el mismo metacaracter \

```
edwar@localhost:~>   echo \ \
\
edwar@localhost:~>
```

Uso de ‘ ‘

```
edwar@localhost:~>  echo ‘Let us get $100 * 20’
Le tus get $100 * 20
edwar@localhost:~>
```

En la cadena anterior, se tienen dos Metacaracteres, $ y *. Usando un par de comillas simples, se ha desactivado su significado especial
también puede hacerse lo siguiente:

```
edwar@localhost:~>  echo Let tus get \ $100 \ * 20
Let us get $100 * 20
edwar@localhost:~>
```

Aquí se emplearon dos Metacaracteres \

Uso de “”

El uso de comillas dobles se explica mejor con el comando date.
Las comillas invertidas `` se usan para obtener la salida del comando que se encuentra dentro de ellas, echo date mostrara date en la pantalla, mientras que echo `date` mostrara la fecha del sistema (la salida del comando date)

```
edwar@localhost:~>  echo date
date
edwar@localhost:~>
```

El comando echo imprime la cadena date y no la salida del comando date

edwar@localhost:~> echo `date`
sab abr 04 17:22:23 VET 2009
edwar@localhost:~>

La salida del comando date se muestra aquí

edwar@localhost:~> echo ‘*Date = `date`’
*Date = `date`
edwar@localhost:~>

Si se usan comillas simples, no se obtiene el resultado requerido. Se debe usar comillas dobles.

edwar@localhost:~> echo “*Date = `date`”
*Date = sab abr 04 17:24:05 VET 2009
edwar@localhost:~>

Aquí se obtiene la cadena *Date impresa junto con la salida del comando date Las comillas dobles (“”) desactivan el significado especial de todos los metacaracteres excepto $, ` y \

La Tabla 1.2 muestra todos los metacaracteres

Categoría	Metacaracter	Descripción
Notaciones de linea de comando	\ (barra invertida)	Especifica que el comando continúa en la siguiente linea. Cuando se ejecuta, se muestra un símbolo > en la pantalla. Esto indica que el comando de la linea precedente no esta completo
	& (ampersand)	Especifica que el comando declarado en esta linea debe ser ejecutado como un proceso de segundo plano
	; (punto y coma)	Separa comandos en la misma linea
	( ) (paréntesis)	Agrupa comandos específicos dentro de ( ) en uno solo
	&& (doble ampersand)	Especifica que el segundo comando se ejecutara solo si el primero lo hace
	| | (doble tubería)	Especifica que para ejecutarse el segundo comando debe fallar el primero
Sustitución de comodín	? (signo de interrogación cerrado)	Cualquier carácter único
	* (asterisco)	Cualquier combinación de caracteres. Esta combinación puede incluir también el carácter null, excluyendo el punto inicial
	[list]	Cualquier carácter contenido en una lista
	[^list]	Cualquier carácter no contenido en una lista
Redirección y Tuberías	> archivo	Hace de archivo la salida estándar. Todas las referencias a la salida estándar ahora se refieren a este archivo
	< archivo	Hace de archivo la entrada estándar. Todas las referencias a la salida estándar ahora se refieren a este archivo
	>> archivo	Hace de archivo la salida estándar, agregando la salida a archivo si ya existe
	<< word	Especifica que la entrada del shell se va a tomar

		hasta la primera linea que contenga word o hasta el final del archivo. Esta característica es útil cuando se hace programación shell. Sin embargo, esta fuera del alcance de esta discusión
	1>&2	Permite redireccionar la salida estándar al error estándar. Cuando las aplicaciones se ejecutan, permite que los mensajes de error se redireccionen al error estándar, en lugar de la salida estándar. Esto ayuda a evitar desorden de la salida y los mensajes de error
	comando 1 | comando 2	A esto denomina tuberías (piping). Hace que la salida estándar de comando 1 sea la entrada estándar de comando 2.
Variables Especiales de Shell	$#	Especifica el numero de argumentos al shell
	$-	Especifica las opciones del shell
	$?	Especifica el estado de la salida del ultimo comando
	$$	Especifica el numero del proceso o el PID del proceso shell actual
	$!	Especifica el numero del proceso o el PID del ultimo comando de segundo plano
	$0	Especifica el nombre del comando que se esta ejecutando actualmente.
	$*	Especifica la lista de parámetros posiciónales. Son los argumentos pasados al shell. Se usa en programación shell.
Sustitución de Parámetros	${variable-word}	Muestra el valor de variable si existe; en caso contrario muestra el valor word
	${variable=word}	Muestra el valor de la variable si este existe; en caso contrario muestra el valor de word y asigna a variable el valor de word
	${variable+word}	Si variable esta asignada, muestra word, en caso contrario no muestra nada
	${variable?mesg}	Si variable ya esta asignada, muestra el valor, en caso contrario muestra mesg y sale del shell
Comillas	\ (barra invertida)	Especifica que el carácter siguiente a \ debe ser interpretado literalmente
	‘ ‘ (comillas simples)	Especifica que el carácter encerrado entre ‘ ’ debe ser interpretado literalmente
	“” (comillas dobles)	Especifica que el carácter siguiente a “” debe ser interpretado literalmente. Las comillas dobles (“”) desactivan el significado especial de todos los metacaracteres excepto $, ` y \, los cuales conservan su función.

Tabla 1.2: Lista de Metacaracteres

Antes de terminar esta unidad, se va a aprender otra característica del shell de Linux, que es la capacidad de completar los nombres de archivos.

4. Completar Nombres de Archivos

Linux proporciona una forma de ayudar a completar un nombre de archivo. Se usa las tecla tab. (Tabulador) para este propósito. Por completar nombres de archivo, se entiende la capacidad de ingresar los primeros caracteres (uno o más) del nombre del archivo y seguir los caracteres con la tecla tab., el shell completa el resto de caracteres del nombre del archivo. Esto es útil cuando los nombres de archivo son largos y ayuda al evitar ingresar los nombres completos. Por ejemplo, si se desea ver el contenido del archivo linuxCont.doc, normalmente se ingresa el nombre completo del archivo. En lugar de esto, si se ingresa una parte de este y se le sigue con la tecla tab., el shell completa el resto, tal como se muestra a continuación.

edwar@localhost:~> cat linuxCo<tab key>
edwar@localhost:~> cat linuxCont.doc

En el ejemplo anterior se asume que el directorio no contiene ningún otro archivo con un nombre similar. Para entender mejor esto, asuma que el directorio actual de trabajo contiene los siguientes archivos y directorios (resaltados en negritas)

linux1.txt
linux2.txt
linuxChap1.doc
linuxChap2.doc
linuxCont.doc
linuxInternals
linux.txt.

Si hay mas de un nombre de archivo con diferentes caracteres después de Co, entonces el sistema los mostrara todos y se podrá elegir de la lista. En el ejemplo, se tiene un solo archivo que tiene el patrón Co. Sea otro ejemplo para entender lo que ocurre cuando hay mas de un archivo que tiene el mismo modelo inicial de caracteres.

Si se ingresa

edwar@localhost:~> cat linux<tab key>

Se observa que el shell no esta llenando ningún carácter. Presionando la tecla tab. (Tabulador) nuevamente, el shell mostrara todos los nombres de archivos empezando por el patrón inicial de caracteres linux. Si se ingresa la tecla tab. nuevamente como se muestra:

edwar@localhost:~> cat linux <tab key><tab key>

linux1.txt linux2.txt linuxChap1.doc linuxChap2.doc
linuxCont.doc **linuxInternals** linux.txt./

Cuando se ingresa cat linux<tab key><tab. key> el shell muestra todos los archivos que empiezan con el patrón linux, como se mostró anteriormente y luego presenta el prompt con cat linux esperando ingresar unos pocos caracteres mas y presionar la tecla tab. nuevamente; el shell puede completar el resto si no hay otros archivos con un patrón similar nuevamente.

La característica de completar nombres de archivo de un shell de Linux es útil por las siguientes razones:

- Cuando los nombres de archivo son muy largos
- Cuando hay un gran numero de archivos en un directorio
- Cuando un gran numero de archivos tienen nombres similares.

Unidad 2: Laboratorio Fundamentos del Shell

Objetivos de Aprendizaje

Al final de esta unida Ud. será capaz de:

- Trabajar con variables shell
- Crear y asignar variables definidas por el usuario
- Trabajar con diferentes metacaracteres del shell

Ejercicios de Laboratorio

1) Crear un directorio y llamarlo Personal en su directorio home
2) Muévase a Personal y cree los siguientes documentos y subdirectorios.
   Documentos
   - Libro1.txt
   - Libro2.txt
   - Album1.txt
   - Album2.txt
   - Pasatiempo.txt
   - Deporte.txt

   Subdirectorios
   - Colecciones
   - Favoritos
   - Álbumes
   - Natural
3) Listar los archivos usando los metacaracteres, tal como:
   ?, *, [list] [^list]
4) Usar todos los metacaracteres de redirección como >, ; &&, | |
5) Crear una variable shell llamada name y asignarle un valor llamado Marcos
6) Usar la sustitución de parámetros para cambiar name de Marcos a Ricardo

Unidad 3: Características del Shell

Objetivos de Aprendizaje

Al final de esta unida Ud. será capaz de:

- Explicar las variables predefinas del shell
- Explicar los parámetros posiciónales
- Describir como trabaja el comando exp.
- Discutir acerca de los comandos read y test
- Describir la familia de filtros grep y filtros sed

1. Introducción

En la Unidad 1 – *Fundamentos del Shell* de este volumen, se presentaron diferentes variables usadas por un shell. Se aprendió de diferentes variables shell, tanto las definidas por el sistema como las definidas por el usuario. En esta unidad, se aprenderán mas detalles acerca del shell y sus características.

Las variables predefinidas son variables definidas por el sistema y se discutirán en esta unidad todas predefinidas del shell. También se aprenderá acerca de los parámetros posicionales. Se manejarán algunos comandos como exp., read y test.

Se explicarán también los filtros con detalle, al igual que algunas facilidades adicionales de shell a través de comandos que tienen una sintaxis similar a C.

2. Variables Shell Predefinidas

Las variables shell predefinidas son variables definidas por el sistema para cada shell, que es el caso contrario a las variables definidas por el usuario. Generalmente, las variables shell, tanto las predefinidas como las definidas por el usuario, pueden ser establecidas de dos formas, asignando un valor a la variable o activando la variable.

variable=value

Establece el valor a la variable

set variable

Activa la variable

Nombre	Valor
BASH	Proporciona un nombre de ruta de una instancia bash, por ejemplo, echo $BASH resultara /bin/bash
BASH_VERSION	Proporciona la versión actual de una instancia de bash, por ejemplo, echo $BASH_VERSION resultara 2.05.8(1) release
COMMAND_ORIENTED_HISTORY	Almacena comandos de linea simple o múltiples en el archivo history
HISTSIZE	Proporciona información acerca del numero de comandos que se van a almacenar en el archivo history, por ejemplo $HISTSIZE resultara en 1000
HOME	Proporciona informa acerca del directorio home del usuario actual, por ejemplo, echo $HOME entrega /home/edwar
HOSTNAME_COMPLETION_FILE	Informa al bash acerca del metodo para completar los nombres de host (hostname). Los nombres host son nombres dados a un computador de una red. Por defecto es /etc/host, que representa un archivo host del sistema.
HOSTTYPE	Proporciona información acerca de la naturaleza del host en que el usuario esta, por ejemplo, echo $HOSTYPE dará la salida i386
MAIL	Notifica al usuario, si hay nuevo correos o no, por ejemplo, mail dará ‘No mail’ para edwar
MAILCHECK	Le indica al bash acerca del intervalo de tiempo en el que debe verificar por nuevos correos
MAILPATH	Proporciona una lista de nombres de archivos en la cual recibir los correos. Estos nombres están separados por dos puntos (:). El nombre por defecto es /var/spool/mail/$USER, donde $USER será sustituido con el nombre login del usuario

NOTIFY	Notifica al usuario acerca de trabajos de segundo plano terminados inmediatamente. En caso contrario, reporta al usuario antes de proporcionar el prompt primario.
OLDPWD	Proporciona información al usuario acerca del reciente directorio de trabajo previo, por ejemplo echo $OLDPWD puede devolver /home, si el directorio de trabajo previo fue /home
PATH	Muestra la ruta de búsqueda del comando, por ejemplo, echo $PATH mostrara /user/kerberos/bin:usr/local/bin:/usr/bin:/usr/X11R6/ bin:/home/edwar/bin
PROMPT_COMMAND	Generalmente, se ejecuta antes del comando prompt primario
PS1	Proporciona información acerca del prompt primario, por ejemplo, echo $PS1 mostrara [\ u@ \h \W]\ $
PS2	Proporciona información acerca del prompt secundario, por ejemplo, echo $PS2 mostrara >
PPID	Proporciona la identificación (ID) del proceso shell padre, por ejemplo, echo $PPID devolverá 26887
PWD	Proporciona información acerca del directorio actual de trabajo, por ejemplo, $PWD devolverá /home/edwar
RANDOM	Genera un nuevo numero aleatorio cuando lo requiera el usuario, por ejemplo echo $RANDOM generara un numero 30482. sin embargo, no se puede asignar este número a una variable al momento de la creación.
SECONDS	Proporciona información acerca de cuando fue invocado el shell por ultima vez en segundos, por ejemplo echo $SECONDS mostrara 385 como resultado.
TMOUT	Especifica el tiempo que el sistema debe esperar por entrada del usuario antes de salir, por ejemplo, $TMOUT 10 le dice al sistema de esperar 10 segundos antes de salir.
UID	Proporciona la identificación del usuario actual. Por ejemplo, echo $UID devolverá 503, si la identificación de usuario asignado al usuario es 503

Tabla 3.1: Variables Shell Predefinidas.

El uso de algunas variables shell predefinidas se presenta a continuación:

Uso de Algunas Variables Shell

edwar@localhost:~> echo $HOME
/home/edwar
edwar@localhost:~>

En este ejemplo, la salida de echo $HOME es la misma que habría sido para pwd ya que el directorio actual de trabajo es el directorio home.

Nota: pwd no es lo mismo que PWD, pwd es el comando y PWD es la variable predefinida shell

edwar@localhost:~> echo $UID
501

edwar@localhost:~>

501 es el ID que se le ha asignado al usuario cuando se creo el login para el mismo

edwar@localhost:~> echo PS1=`date`$
lun abr 06 12:28:36 VET 2009$

Que le ha ocurrido al prompt de comandos edwar@localhost:~> Se ha cambiado la cadena del prompt de edwar@localhost:~> a lun abr 06 12:28:36 VET 2009$ asignado el resultado del comando date. PS1 es una variables shell que permite a los usuarios cambiar la cadena del prompt a cualquier otra cadena que se desee.
Se ha asignado la salida del comando date seguido por $ a PS1
Nota: encerrando la fecha dentro de comillas invertidas se obtiene la salida del comando date.
Para cambiar la cadena del prompt a la original, se puede simplemente reasignar PS1

lun abr 06 12:28:36 VET 2009$ PS1=PWD
/home/edwar

Ahora se ha reasignado edwar@localhost:~> como la cadena del prompt. Aquí se ha usado PWD como directorio actual de trabajo, que es /home/edwar.

Fin del Uso de Algunas Variables Shell

Se ha aprendido una característica muy importante del shell de Linux, que permite a los usuarios cambiar y personalizar cadenas de prompts. En lo que queda del curso se usara la cadena del prompt /home/edwar$ estableciendo la cadena prompt PS1 a PWD, como se mostró en el ultimo ejemplo. Sin embargo, este cambio debe hacerse para cada sesión de ingreso (login)

En la Unidad 6 : *Personalización del Ambiente del Usuario*, de este volumen, se aprenderá como realizar personalizaciones adicionales, las cuales no requieren cambiar la cadena del prompt durante cada sesión de ingreso (login).

PS2 es la segunda cadena prompt. La segunda cadena prompt por defecto es >, la cual se mostró cuando se uso el metacaracter de continuación de linea (\). A PS2 también se le puede asignar cualquier cadena elegida por el usuario.

3. Parámetros Posicionales

Cada shell proporciona un lenguaje llamado *script shell* o *programa shell*, el cual se puede usar para escribir programas. El archivo que maneja el script shell se denomina *shell file*.

Se pueden pasar argumentos de linea de comandos a un archivo shell. Estos argumentos están contenidos en variables shell que son referidos como parámetros posicionales.

Las siguientes son algunas de las características de los parámetros posicionales:

- Son referenciados con las variables $0, $1, $2, etc. $0 contiene el nombre del Script, el primer argumento esta contenido en $1, el segundo en $2 etc.
  Nota: los nombres de las variables no son 1, 2, 3 y etc. Estos son solo notaciones compactas para indicar los parámetros posicionales de un archivo shell
- El digito siguiente al símbolo $ indica la posición en la linea de comando.
- A diferencia de las variables de un lenguaje de programación regular, los parámetros posicionales o variables de argumentos, como se denominan, no pueden ser alterados.

4. El Comando expr

El comando expr se presenta a continuación para ilustrar como se usan los parámetros posicionales. El comando expr evalúa la expresión dada como argumento e imprime el resultado en la salida estándar.

Uso del Comando 3.1

/home/edwar$ expr 2 + 3 + 5
10
/home/edwar$

Los operadores que son permitidos en las expresiones proporcionadas a expr son |, &, !=, =, <, >, >=, +, -, *, / y %. Como *, > y < son metacaracteres, cuando estos operadores se usan en el comando expr, deben combinarse con el carácter \ para ignorar el significado especial de dichos caracteres. Véase el ejemplo dado a continuación:

```
/home/edwar$ expr 2 \* 3
6
/home/edwar$
```

Si no se hubiera usado el operador \, el comando habría interpretado * como todos los archivos del directorio y los mostraría. Para los operadores <= y >= también debe usarse \ porque el shell los interpretara de otra forma a > y < como caracteres especiales

Fin de Uso de Comando 3.1

Adicionalmente, es posible emplear las expresiones que se muestran en la Tabla 3.2, las cuales manejan básicamente cadenas. Las entradas de la tabla son de las páginas man de SuSe Linux

Expresiones	Descripción
match STRING REGEXP	Compara la cadena STRING con una expresión regular o con otra cadena
substr. STRING POS LENGTH	Proporciona la subcadena de STRING. La posición POS se cuenta a partir de 1
index STRING CHARS	Proporciona el índice en STRING donde cualquier CHARS se encuentra. Si no se encuentra CHARS, devuelve 0.
length STRING	Proporciona la longitud (tamaño) de STRING
+ TOKEN	Interpreta TOKEN como una cadena incluso si se usan las palabras claves como ‘match’ o el operador ‘/’.

Tabla 3.2: Operadores para expr y descripciones

Las palabras que se muestran a continuación en negrita son los argumentos de expr, basados en los cuales ejecuta ciertas acciones.

Uso de Comando 3.2

```
os=linux
/home/edwar$ expr match linux $os
5
/home/edwar$
```

Compara la cadena linux con el contenido de la variable os

```
/home/edwar$ expr substr hello 3 5
llo
/home/edwar$
```

Imprime la subcadena que empieza en la posición 3 de la cadena

```
/home/edwar$ expr index hello e
2
/home/edwar$
```

Imprime la posición de la letra e en la cadena

```
/home/edwar$ expr index  hello eo
2
/home/edwar$
```

Imprime la posición de la primera letra encontrado en la cadena, que es e nuevamente.

```
/home/edwar$ expr length hello
5
/home/edwar$
```

Imprime el numero de caracteres presentes en la cadena

/home/edwar$ expr **index** match m
expr: error de sintaxis
/home/edwar$

Devuelve error, ya que match es un comando expr

/home/edwar$ expr **index** + match m
1
/home/edwar$

Imprime la posición de la letra m en la cadena match. En este caso se le indica a expr que interprete a match como una cadena simple. Usando +, los comandos para expr regresan a su significado original.

Fin de Uso de Comando 3.2

Luego se tocaran aspectos acerca de otros comandos que se pueden ejecutar en el shell y usar en scripts shell.

5. El Comando find

Este comando se usa para buscar archivos en la jerarquía de directorios. La sintaxis para esta comando se muestra a continuación:

find [ruta] expresión.

En el ejemplo anterior, ruta especifica el directorio a partir del cual la búsqueda debe empezar. Para comprender este comando, se asume que el directorio home tiene los siguientes archivos:

archivo1.txt
nuevoarchivo.txt
passwd

Uso de Comando 3.3

/home/edwar$ find –name passwd –print

En el ejemplo anterior, se ha proporcionado un patrón passwd seguido de la opcion –name. Lo que sigue a –name se trata como un nombre de archivo y se busca la ruta especificada por ruta. No se ha especificado ninguna ruta, así que solo se busca y muestra el directorio actual. La opcion –print causa que la ruta completa sea impresa.

/home/edwar$ find / -name passwd -print
/etc/passwd
/usr/bin/passwd
/usr/local/ftp/etc/passwd
/home/edwar/passwd
/home/edwar$

El único cambio en el find anterior es que la búsqueda se inicia en otro punto de la jerarquía del directorio. Se especifico /, significando la raíz. El comando busca en la jerarquía completa de la estructura del directorio y muestra la ruta completa del nombre de archivo cuando lo encuentra en un directorio. No se puede buscar en aquellos directorios que no tienen permisos de ejecución. El siguiente mensaje se muestra para todos esos directorios:

find:/usr/lib/irdc: Permiso denegado.

Esta disponible un numero de opciones para expresión utilizadas con el comando find. Para mayor información acerca del comando, refiérase a las paginas man de find

Fin de Uso de Comando 3.3

6. El Comando read

El shell proporciona un comando read incorporado que lee una linea de texto de la entrada estándar y asigna el valor (en otra palabras, el texto) a una variable con nombre. Sin embargo, el valor del texto asignado no tiene el carácter de una linea.

Entre los usos del comando read se destacan: configuración de variables de entorno en .bashrc, asignación de valores de variables en programas Shell Scripts, entre otros.

Se aprenderá acerca de .bashrc y programas de shell scripts en la Unidad 6 – *Personalización del Ambiente de Usuario* de este volumen.

Uso de Comando 3.4

/home/edwar$ read firstVar
hello <enter>
/home/edwar$

La cadena hello se almacena en la variable denominada firstVar. La cadena 'hello' no es la salida del comando read, sino la entrada dada por el usuario. El argumento para read es el nombre de la variable. El comando read lee de la entrada estándar.

/home/edwar$ read secondVar
my world
/home/edwar$

Se ha asignado también una cadena secondVar.

/home/edwar$ echo $firstVar $secondVar
hello my world
/home/edwar$

Imprime el contenido de ambas variables

Fin de Uso de Comando 3.4

7. El Comando test

El comando test en Linux se usa para realizar las siguientes funciones:

- Verificar si el argumento dado es un archivo o un directorio
- Determinar si se han pasado suficientes argumentos para los scripts shell
- Comparar cadenas de caracteres o números dados

El comando test simplemente devuelve un estado de la salida al shell, reportando éxito o error.

Uso de Comando 3.5

/home/edwar$ flower=rose
/home/edwar$ test $flower = rose
/home/edwar$

Se asigno la variable definida por el Usuario flower el valor rose. A continuación, se prueba usando el comando test si la variable flower realmente tiene el valor rose. No hay salida desde test. Devuelve un estado de salida que puede ser capturado en comandos como while, until e if en un script shell. Algunas de las opciones que el comando test puede tomar son –d –w y –x.

/home/edwar$ test –d programas
/home/edwar$ echo $?
0
/home/edwar$

El estado de salida del último comando ejecutado es 0, lo cual indica éxito. -w y –x indican si el argumento (un archivo o directorio) es de escritura y ejecutable respectivamente. Esto depende de los permisos asignados por el usuario. El comando test es muy útil en programación shell para realizar acciones basadas en las condiciones.

Fin de Uso de Comando

Una vez estudiado el tema acerca de las variables predefinidas, los parámetros posicionales y como trabajar con expr, test y los comandos read, se aprenderá acerca de grep y sed.

8. Filtros en Linux

Ya se ha explicado acerca de los filtros Linux en el Volumen 1, Unidad 2: *El Sistema Linux*. En esta unidad, se aprenderá acerca del uso de grep, la familia grep de filtros y el filtro sed. Para entender estos poderosos filtros, se debe comprender primero las

expresiones regulares.
Una expresión regular es la descripción de una plantilla, la cual se usa para comparar contra un cadena de caracteres. Las expresiones regulares son formulas que hacen corresponder una cadena con un patrón dado. El patrón usa caracteres que tienen un especial significado. La expresión regular se especifica como una cadena.
Las expresiones regulares se usan para búsquedas sensitivas al contexto y modificación de texto. Proporcionan un metodo para seleccionar una cadena específica de un conjunto de cadenas.
Algunos caracteres especiales usados en expresiones regulares se muestran en la Tabla 3.3. Sin embargo, esto no es una lista exhaustiva.
Se aprenderá mas acerca de los caracteres especiales usados en las expresiones regulares cuando se aprenda acerca de la familia de comandos grep y el filtro sed.
grep y sed tienen su propio conjunto de metacaracteres.

Carácter Especial	Significado
. (punto)	Equivale a cualquier carácter único. r. n coincidirá con run, ran, rbn, ron etc,. **Nota:** el . (punto) significa un solo carácter entre r y n. La linea puede tener caracteres antes de r y después de n. Así el patrón relaciona abrun, ranbb o abrnakk
[ ] (corchetes)	Equivale a cualquier carácter especificado en el rango. [1234] book coincidirá con 1book, 2book, 3book, 4book
[^]	Equivale a cualquier carácter que no este en el rango [^abc] equivale a todas las líneas que contienen cualquier carácter que no este en [a, b, c]. Así adf y kkkkc coincidirán, pero no aabc o aaaa
?	Equivale a cero o una ocurrencia del carácter precedente, ab? Coincidirá con a y b
* (asterisco)	Equivale a cero o mas ocurrencias del carácter precedente, ab* equivale a, ab, abb, abbbb, aaa, etc, aaa y acc también coinciden porque el * relaciona cero o mas ocurrencias del carácter precedente, que es b en este caso.
+	Equivale a una o más ocurrencias del carácter precedente, ab+ coincide con ab, abb, abbbb, etc,. No coincide con aaa y acc ya que + indica una o mas ocurrencias del carácter precedente de b.
^ (acento circunflejo)	Equivale al inicio de la linea. Por ejemplo, ^this coincide con todas las líneas que tienen this como patrón de inicio de líneas.
$	Equivale al final de la linea. Por ejemplo, $this coincide con todas las líneas que tienen this como patrón de finalización de líneas.

Tabla 3.3: Metacaracteres Usados con Expresiones Regulares

La Tabla 3.3 proporciona una lista de algunos metacaracteres simples. Sin embargo, no todos ellos se usan con los comandos de búsqueda de patrón, grep y sed.

8.1 grep, egrep y fgrep

grep siglas en ingles de **g**lobal **r**egular **e**xpression **p**rint (impresión global de expresión regular). Fue presentado en el volumen 1, Unidad 2 –*El Sistema Linux*. El uso mas simple de grep se muestra a continuación:

grep patrón nombrearchivo

A continuación se aprenderá más acerca de grep y su familia de filtros. El patrón para grep pueden ser los metacaracteres ^ y $, que se usan para fijar el patrón al inicio o fin de la linea, respectivamente. Considere un archivo denominado filtros.txt, para comprender alguno de los metacaracteres que grep usa. Asuma que este archivo tiene las siguientes líneas:

Los filtros son programas que leen la entrada, la procesan y
la escriben en la salida. Algunos ejemplos de filtros
son sort, tail, head y grep. Aquí tenemos más
filtros en Linux. grep es un poderoso filtro
que busca patrones específicos y los imprime en pantalla.

Uso de Comando 3.6

/home/edwar$ grep filtros filtros.txt

Los filtros son programas que leen la entrada, la procesan y
la escriben en la salida. Algunos ejemplos de filtros
filtros en Linux. grep es un poderoso filtro

/home/edwar$

El uso anterior de grep es el más sencillo. Encuentra el patrón filtros en el archivo filtros.txt y muestra las líneas que coinciden con este.

/home/edwar$ grep ^filtros filtros.txt

filtros en Linux. grep es un poderoso filtro

/home/edwar$

Muestra solo las líneas que tienen el patrón filtro al inicio de la linea

/home/edwar$ grep filtros$ filtros.txt

la escriben en la salida. Algunos ejemplos de filtros
filtros en Linux. grep es un poderoso filtro

/home/edwar$

Muestra solo las líneas que tienen el patrón filtro al final de la linea

/home/edwar$ ls –l | grep ‘^d’

Imprime lo nombres de los subdirectorios del directorio de trabajo actual. Se obtiene los nombres de los subdirectorios como resultado del comando anterior, esto se debe a que el listado completo muestra los permisos de los archivos y si el archivo es un directorio, la linea comienza con el carácter d. El comando anterior imprimirá las líneas que contienen el carácter ‘d’. Las comillas simples se usan para quitar el significado especial de ^.
También se puede usar [list] y [^list] en grep. Por ejemplo considere lo siguiente:

/home/edwar$ ls –l | grep [5-6]

Equivale a todas las líneas que tienen 5 o 6 en ellas. Las demás son ignoradas.

Se aprendió que el . (punto) equivale a un solo carácter. * (asterisco) se aplica al carácter previo o metacaracter en la expresión y hace coincidir cualquier numero de ocurrencias sucesivas del carácter o metacaracter, por ejemplo:

/home/edwar$ ls –l | grep f

Esto equivale a todas las líneas que tienen por lo menos una ‘f’ en ellas e ignora aquellas que no. Se obtiene el resultado como se declaro, siempre que exista al menos un archivo en el directorio que tiene f en su nombre. En caso contrario no muestra nada.

Fin de Uso de Comando 3.6

Egrep y fgrep son versiones un poco mas especializadas de grep. Ambas pueden aceptar un archivo comando con la opcion –f. Además, egrep puede usar el operador | y ( ) para agrupar expresiones.

La búsqueda de patrones usando egrep y fgrep es más rápida, porque se lleva a cabo en paralelo. Los siguientes puntos muestran la forma en que se hace la búsqueda de patrones usando egrep y fgrep.

- Usando () con egrep, se puede tener un patrón (ab)* que equivale a ab, abab, ababab y así sucesivamente.
- egrep puede usar a+ a?. grep no permite el uso de + y ?
- a+ equivale a una o mas ocurrencias de a y a? equivale a cero o una ocurrencia de a.
- a* en grep, egrep y fgrep se refiere a cero o mas ocurrencias de a

Uso de Comando

/home/edwar$ egrep ‘(the)’ + lineas.txt

Muestra todas las líneas del archivo lineas.txt que tienen al menos un patrón the. + significa una o mas ocurrencias de the.

/home/edwar$ egrep ‘(t|s)he+’ lineas.txt

Equivale a todas las líneas que tienen t o s antes del patrón he. El operador | se usa para denotar o.

Fin de Uso de Comando 3.7

Uso de Comando 3.8

Asuma que el archivo idnos contiene el numero de identidad de los estudiantes como se muestra a continuación.

/home/edwar$ cat > idnos

101
102
103

<ctrl + d>

/home/edwar$

El archivo detallesestudiantes contiene los siguiente.

/home/edwar$ cat > detallesestudiantes

101 Juan
102 Maria
103 Ricardo

<ctrl + d>

/home/edwar$

Usando fgrep se pueden encontrar las líneas que coinciden con los patrones, proporcionados en el archivo de entrada

/home/edwar$ fgrep –f idnos detallesestudiantes

101 Juan
102 Maria
103 Ricardo

/home/edwar$

La opcion –f indica a fgrep de buscar patrones en un archivo, idnos contiene una lista de números de identidad de estudiantes de una clase. fgrep encuentra las cadenas de patrón de búsqueda en el archivo idnos. Usando los patrones en el archivo idnos, fgrep busca en el archivo detallesestudiantes todos estos patrones.

Fin de Uso de Comando 3.8

8.2 sed

sed es el editor de flujos derivado del editor ed. sed también funciona como un filtro.

La sintaxis de sed se muestra a continuación:

/home/edwar$ sed ‘lista de comandos’ nombres de archivos

sed lee una linea a la vez de los archivos de entrada, aplica la lista de comandos y muestra las líneas en la salida estándar. Los comandos ed forman la lista de comandos para sed. sed no altera el archivo original. Por medio de sed, se pueden mostrar las líneas que se desea, borrarlas, sustituirlas con otro texto, etc. El archivo intermedio procesado puede ser redireccionado a otro archivo.

sed puede realizar las siguientes funciones:

- Aceptar como entrada una combinación de números de linea y un comando
- Aceptar como entrada una combinación de patrón y comando
- Sustituir una cadena antigua con una nueva

Uso de Comando 3.9

/home/edwar$ sed ‘2q’ filtros.txt

Sale después de mostrar 2 líneas, ya que 2 indica el numero de líneas y q es el comando para salir

/home/edwar$ sed ‘$p’ filtros.txt

Imprime todas las lineas

/home/edwar$ sed ‘/the/!d’ filtros.txt

El comando anterior muestra las líneas que contienen la cadena de caracteres the, en el archivo filtros.txt

/home/edwar$ sed ‘/^[\t] * $/d’ filtros.txt

Borra todas las líneas en blanco del archivo. Este es un ejemplo de un patrón y un comando. El patrón es un blanco al inicio (^), seguido de un blanco o un tabulador ([\t]), seguido por cualquier numero de ocurrencias del metacaracter previo (*) hasta el final de la linea ($). El patrón ^[\t]*$ significa una linea que contiene solo blancos o tabs. Sed borra todas estas lineas muestra el resto del contenido del archivo.

La forma general de sustitución es:

/oldstring/newstring

oldstring y newstring pueden ser cadenas simples.

Fin de Uso de Comando 3.9

Unidad 4: Laboratorio Características del Shell

Objetivos de Aprendizaje

Al final de esta unida Ud. será capaz de:

- Trabajar con variables shell predefinidas
- Aplicar los comandos read y test basados en la necesidad de una solución a un problema
- Resolver problemas usando grep, egrep, fgrep y filtros sed

Ejercicios de Laboratorio

1) Conectarse al sistema y determinar su UID
2) Existen dos formas de encontrar la ruta del directorio de trabajo. Use ambos métodos
3) Cambiar el prompt de la cadena actual al directorio actual de trabajo seguido de la fecha y del signo peso ($)
4) Cambiar la segunda cadena del prompt al carácter # y verificar usando el metacaracter de continuación de linea
5) Mostrar el contenido de la variable definida por el usuario llamada var. El valor en var debe asignarse por medio del comando read
6) Determinar si un subdirectorio tiene permiso de escritura
7) Crear un archivo llamado miarchivo.txt e ingresar el siguiente texto en el:

    Hang over us
    Come over
    Stay over
    Dine with us and hang over with others
    Let us enjoy and stay over with us for the day
    Play with us
    Let us party

8) Mostrar las lineas que contengan Hang over o Stay over
9) Mostrar las lineas que tienen el carácter s seguido por un espacio
10) Usar sed para sustituir todas las ocurrencias de la palabra over con la palabra above
11) Probar todos los comandos, metacaracteres y variables shell que se ha aprendido hasta ahora

Unidad 5: Procesos en Linux

Objetivos de Aprendizaje

Al final de esta unida Ud. será capaz de:

- Definir un proceso y sus usos en un sistema operativo
- Discutir el comando ps
- Discutir los procesos padre e hijo
- Explicar el concepto de procesos de primer y segundo plano
- Discutir como manejar el ambiente de procesos
- Explicar acerca del proceso demonio
- Explicar acerca de las señales y su importancia en Linux.

1. Concepto de Proceso.

En unidades anteriores, se ha trabajado con numerosos comandos de Linux. Cada comando (programa) que un usuario ejecuta en Linux es una instancia de ese programa. Este concepto de 'instancia' se entiende claramente con la ayuda de ejemplos simples.

- El usuario edwar esta conectado. Ejecuta el comando date. Repite el proceso después de algún tiempo. Así, el puede ejecutar el comando date cualquier numero de veces durante una sesión.
- Cuatro usuarios, usara, Serv., husera y uzead, sentados en diferentes terminales están conectados a un sistema Linux. Todos los usuarios han ejecutado el programa vi, para crear o editar un archivo de su elección. El sistema Linux proporciona solo una copia del programa vi en el sistema. Pero en el ejemplo, se observa que todos los cuatro usuarios están usando el programa simultáneamente.

En el primer ejemplo, un usuario ejecuta el mismo comando date varias veces. En el segundo ejemplo, cuatro usuarios ejecutan vi al mismo tiempo. Cada vez que se ejecuta un programa, se corre una 'instancia' de ese programa. Esta característica disponible en un sistema multitarea, es posible a traves del concepto de *proceso* en un sistema operativo.

Un proceso (a veces llamado tarea) se define como 'una instancia de un programa en ejecución'

A continuación se presentan algunos puntos importantes acerca de los procesos:

- Un proceso es una entidad básica programada en un procesador por el sistema operativo.
- Un proceso es una entidad dinamica.
- Cada proceso se ejecuta independientemente de otros procesos en el sistema.
- Los procesos también pueden interactuar con otros procesos por medio de un mecanismo permitido por el sistema operativo llamado Comunicación entre Procesos (Inter Process Communication – IPC). En este curso, no se ahondara en detalles del IPC.
- Cuando los procesos comparten datos, el sistema operativo hace uso de la sincronización. La sincronización es un metodo por el cual el sistema operativo asegura que cuando un recurso compartido siendo usado por un proceso, los otros procesos no tienen acceso a el.

Se discutió que el sistema operativo permite que cuatro usuarios trabajen con el programa vi, cada uno ejecutando una instancia de vi. Aunque la invocación a vi puede ser simultanea, la maquina en la que el sistema operativo reside tiene un procesador para manejar cuatro solicitudes. Así, es importante para el procesador, administrar la ejecución de programas solicitados por múltiples usuarios conectados al sistema simultáneamente. El kernel lo permite, al usar el principio de tiempo compartido. Cambia de un proceso a otro de manera que los usuarios no sienten ningún retraso.

En cualquier momento, cuando múltiples procesos están activos, se le asigna a un proceso el tiempo de CPU. Este proceso se dice que esta en estado de 'ejecución' mientras los otros están en estado de 'suspensión' (sleeping). Es posible averiguar acerca de todos los procesos que están actualmente activos en el sistema por medio del comando ps.

2. El comando ps.

Se puede conocer el estado de un proceso usando el comando ps, donde ps equivale a *process status* (estado del proceso)

Uso de Comando 5.1

```
/home/edwar$ ps
PID         TTY         TIME        CMD
734         pst/1       00:00:12    bash
754         pts/1       00:00:01    ps
/home/edwar$
```

PID muestra el *número de identificación* del proceso. Cada proceso obtiene un numero de identificación único asignado por el kernel
TTY es el *tipo de terminal* asociado con el proceso. También puede ser otro dispositivo de entrada conectado al sistema.
TIME es el tiempo que el proceso ha estado en ejecución hasta ahora
CMD es el nombre del comando que esta siendo ejecutado, cuyo reporte de estado se esta visualizando. Bash es el proceso shell.

```
/home/edwar$ ps –A
PID         TTY         TIME        CMD
26887       pts/6       00:00:00    bash
26951       pts/6       00:00:00    bashop
27104       pts/6       00:00:00    bash
29214       pts/6       00:00:00    eth0
32067       pts/4       00:00:00    bash
32234       pts/8       00:00:00    bash
309         pts/3       00:00:00    bash
367         pts/9       00:00:00    bash
746         pts/3       00:00:00    ps
/home/edwar$
```

La opcion –A muestra los otros procesos ejecutándose en el sistema. El comando básico ps solo muestra la ejecución de los procesos del usuario actual.

```
/home/edwar$ ps r
PID         TTY         STAT        TIME        CMD
752         pts/3       R+          0:00        ps r
/home/edwar$
```

La opcion –r muestra los detalles para el proceso ejecutándose actualmente.
También se puede ver un listado largo del comando ps, usando la opcion –l. El comando ps –l proporciona el siguiente reporte de estado:

```
/home/edwar$
F    S    UID   PID   PPID  C    PRI   NI   ADDR        SZ    WCHAN
100  S    507   309   308   0    72    0    --------    609   wait4
TTY  TIME CMD
pts/3 00:00 bash
/home/edwar$
```

Se nota que hay muchas columnas; algunas de las mas importantes se estudian en la siguiente sección

3. Proceso Padre e Hijo

En los ejemplos anteriores sobre el comando ps, se nota que el comando mismo ps se muestra en la salida. La siguiente ejecución de ps resultara en PID para ps, como se muestra:

```
 /home/edwar$ ps
       PID         TTY         TIME        CMD
       734         pts/1       00:00:12    bash
       781         pts/1       00:00:02    ps
/home/edwar$
```

Se nota que el PID para el shell permanece, pero el PID para el comando ps difiere de los

anteriores. Como se menciono anteriormente, a cada proceso se la ha dado un nuevo PID cuando es ejecutado.
El comando bash es el proceso shell que se esta ejecutando. Esta activo mientras el usuario este conectado, y por lo tanto, se ve que su PID permanece constante. El proceso shell es llamado proceso *padre.* Todos los comandos que corren dentro del shell son los procesos hijos del proceso shell. Así, todas las invocaciones del comando ps serán ejecutados como procesos hijos.
Usando ps, también se puede ver el PID del proceso padre.

```
/home/edwar$
F       S       UID     PID     PPID    C       PRI     NI      ADDR        SZ      WCHAN
100     S       507     309     308     0       72      0       --------    609     wait4
TTY     TIME    CMD
pts/3   00:00   bash
/home/edwar$
```

Se van a explicar algunas de las columnas de la salida. S proporciona el estado de del proceso; es decir, si el proceso esta suspendido (S) o ejecutándose(R). UID es el ID del usuario, PPID es el PID del padre.
Como el sistema de archivos, los procesos también están organizados en una jerarquía. A medida que se producen más procesos, se crean más procesos hijos. Se puede concebir una jerarquía de procesos encadenados juntos por PID y PPID. A continuación un ejemplo para ilustrar esto.

```
/home/edwar$ bash
/home/edwar$ ps –l
F       S       UID     PID     PPID    C       PRI     NI      ADDR        SZ      WCHAN
100     S       507     309     308     0       72      0       --------    609     wait4
000     S       507     722     309     0       75      0       --------    631     wait4
TTY     TIME    CMD
pts/3   00:00   bash
pts/3   00:00   bash
/home/edwar$
```

El comando bash crea un proceso shell hijo dentro del proceso padre shell padre. Aunque ambos se denominan bash, ellos se diferencian por sus PIDs. El comando ps –l ahora se ejecuta en el proceso shell hijo. El PPID del segundo bash es 309 y el padre de ps es 722, que es el PID del segundo bash. Así, se puede ver una jerarquía de procesos.
Siempre que un comando se ejecuta en un shell, temporalmente se crea un nuevo shell y el comando se ejecuta en este shell. Una vez que la ejecución del comando se completa, el control regresa al shell padre, que creo el proceso shell hijo para ejecutar el comando. El shell padre ahora esta listo para continuar.
Dado que se ha creado otra sesión bash, es necesario desconectarse dos veces del sistema. La primera desconexión lleva al bash padre y la segunda fuera del sistema.

4. Proceso Shell

Cuando un usuario se conecta, el kernel empieza un proceso shell para el usuario. Cuando el usuario ingresa un comando, el shell usa el kernel para iniciar un proceso hijo. El comando se ejecuta y el control regresa al shell. El shell entonces espera por más entradas del usuario. Si el usuario desea terminar un proceso que durante algún tiempo ha estado ejecutándose, puede usar el comando kill (matar). El comando kill toma el PID como argumento de la linea de comandos.
A veces el comando kill puede no terminar un proceso porque es posible para el proceso ignorar todas las señales enviadas para terminarlo por el comando kill. Si esto ocurre, un administrador o el propietario del proceso puede usar la opcion -9 con el comando kill. La señal 9 no puede ser ignorada por ninguno de los procesos en ejecución, asi terminara definitivamente el proceso.

Se entenderá el uso de la opcion -9 a traves de un ejemplo.

Uso de Comando 5.2

Asumiendo que lo siguiente es la salida del comando ps, se intentara matar uno de los procesos en ejecución.

```
/home/edwar$ ps
      PID        TTY        TIME       CMD
      970        pts/3      00:00:00   bash
      1072       pts/3      00:00:00   bash
      1103       pts/3      00:00:00   ps
/home/edwar$ kill 1072
/home/edwar$
```

Si se ingresa ps nuevamente, probablemente se vera una salida similar a la anterior, ya que el comando kill no ha terminado el segundo proceso bash.

```
/home/edwar$ ps
      PID        TTY        TIME       CMD
      970        pts/3      00:00:00   bash
      1072       pts/3      00:00:00   bash
      1103       pts/3      00:00:00   ps
/home/edwar$ kill -9 1072
killed
/home/edwar$ ps
      PID        TTY        TIME       CMD
      970        pts/3      00:00:00   bash
      1103       pts/3      00:00:00   ps
/home/edwar$
```

Ahora se observa que el proceso bash hijo ha muerto. Esta es otra forma de salir del programa del proceso bash hijo.

Fin de Uso de Comando 5.2

A continuación se aprenderá acerca de los procesos de primer plano.

5. Proceso de Primer Plano

Se ha visto que cuando se ingresa cualquier comando en el prompt del shell, se crea un nuevo proceso shell. En la instancia de cal, el comando cal será ejecutado bajo este nuevo shell. Este proceso de cal estará ejecutándose en primer plano (foreground). El control no se dará al shell padre hasta la ejecución del comando cal. El usuario no puede realizar ninguna otra actividad. En términos simples, el prompt $ no se mostrara hasta que el programa se complete. Después de la ejecución, termina el shell recientemente creado y shell padre continúa.

Se va asumir que se tiene un programa llamado miPrograma, el cual toma un tiempo demasiado largo para completarse. Cuando se invoca este programa, el shell actual creara un proceso shell hijo en el que se ejecutara miPrograma y también continuar con otro trabajo en el sistema, Linux proporciona un metodo para ejecutar comandos en segundo plano (background).

6. Proceso de Segundo Plano

Los procesos en ejecución en un segundo plano (background) son muy útiles en muchas situaciones. El único cambio que debe hacerse es agregar el símbolo & (ampersand) al final del comando, tal como se muestra a continuación.

```
/home/edwar$ miPrograma &
824
/home/edwar$
```

Al incluir el símbolo & con miPrograma, inmediatamente se presenta el prompt de comandos después de mostrar el PID del proceso que corre en segundo plano. El prompt

de comandos /home/edwar$ indica que se ha regresado al shell padre. Esto significa que ahora se puede continuar trabajando bajo este shell. El & al final del comando indica que el comando debe ejecutarse como un proceso en segundo plano.

6.1 Filtros y Procesos Segundo Plano

```
/home/edwar$ cat miarchivo | wc &
        [1]
/home/edwar$
```

El comando anterior crea dos procesos para cat y wc que se ejecutan en segundo plano. Sin embargo, en este caso, en lugar de imprimir dos PIDs, solo se imprime uno. Este PID impreso es para el último proceso en la secuencia en la linea de comandos, en este caso wc. El PID 1319 se refiere al comando wc. Si se desea terminar el proceso de segundo plano, quizás después de esperar un largo tiempo y no ver resultados, se puede simplemente ingresar el siguiente comando

```
/home/edwar$ kill -9 1319
/home/edwar$
```

Puede haber instancias en las que se tiene que trabajar con programas que toman mucho tiempo en completarse. Suponga que se invoca al programa y luego sale del sistema, Linux inmediatamente termina el proceso. Idealmente, se puede desear que el programa permanezca ejecutándose aun después de haberse desconectado. Esto puede lograrse al usar el siguiente comando:

```
/home/edwar$ nohup miPrograma &
[1] 1396
/home/edwar$ nohup: appending output to `nohup.out`
```

Usando este comando, el proceso para miPrograma continuara ejecutándose incluso después de que el usuario se haya desconectado. (nohup significa ‘no hang-ups’). La salida de miPrograma se guardara en un archivo llamado nohup.out. Los comandos que deben ser iniciados de esta forma, deben ser decididos antes de usar nohup. No se puede usar el comando nohup después de que un comando ha sido inicializado.

Si miPrograma toma mucho tiempo en ejecutarse y los recursos del computador están siendo compartidos por otros usuarios, se puede hacer que este proceso se ejecute con una prioridad baja. En otras palabras, solo se desea ser amable con los otros usuarios usando el siguiente comando:

```
/home/edwar$ nice miPrograma &
[1] 1401
/home/edwar$
```

Establecer nohup automáticamente invocara nice para que haga su trabajo que el proceso se ejecute con una prioridad mas baja.

Linux también permite iniciar un programa en un momento particular.

```
at 6am < miarchivo.
```

Los comandos van a ser almacenados en el archivo miarchivo. Este es redireccionado al comando at. Los comandos del archivo se ejecutaran a las 6 AM.

El comando at puede darse con el tiempo en formato de 24 horas o como una hora especificada, con AM o PM. Este comando se ejecutara como un proceso de segundo plano a la hora señalada.

En este punto, es útil entender que los usuarios pueden averiguar lo siguiente acerca de un proceso.

- El estado de la salida del ultimo comando ejecutado usando $?
- El PID del shell actual usando $$
- El PID del ultimo comando ejecutado en segundo plano usando $!

Se darán detalles en la siguiente sección.

6.2 Variables de Entorno

Se sabe que anteponiendo a una variable el símbolo $ se obtiene el valor de la misma.

Las variables usadas anteriormente son ?, $ y !. Estas son las variables de entorno. Las variables de entorno son variables que mantiene el shell. Estas pueden diferir del shell a shell. Normalmente, se usan para configurar programas utilitarios en un sistema como lpr (se refiere a la impresión fuera de linea) y mail. Establecer estas variables como variables de entorno implica que no se tiene que configurar ciertas opciones cada vez que se conecta al sistema.

Uso de $?, $$ y $!

```
/home/edwar$ cat linux.txt
        Linux es un sistema operativo
        Sigue los estándares GNU.
        Linux esta provisto de un lenguaje de comandos
        El kernel de Linux es el núcleo del sistema operativo
        Linux es un poderoso y versátil sistema operativo
/home/edwar$ echo $?
0
/home/edwar$
```

El comando anterior tuvo éxito, aquí $? mostró 0, que es el estado de salida del comando anterior.

```
/home/edwar$ echo $$
2158
/home/edwar$
```

Se sabe que el shell también es solo otro programa. Esto significa que tiene un PID. $$ proporciona el PID del shell actual, el cual es 2158

```
/home/edwar$
[1] 2166
/home/edwar$ echo $!
2166
/home/edwar$
```

$! Proporciona el PID del ultimo comando ejecutado en segundo plano

Fin de Uso de $?, $$ y $!

7. Administrar el Ambiente de Procesos

Linux proporciona un alto grado de flexibilidad para adaptar el ambiente de procesos, de forma que se ajuste a las necesidades de los usuarios. Por ejemplo, el usuario puede desear cambiar los caracteres asignados para borrar un carácter o borrar una linea, puede desear modificar el contenido de la variable de ruta o cambiar el tipo de terminal.

7.1 Cambiando los Caracteres Erase y kill

Diferentes implementaciones y estándares de los sistemas operativos permiten diferentes caracteres ‘erase’. Un erase de carácter, cuando se usa, borra los caracteres de la pantalla. Si se esta acostumbrado a otro carácter, se puede ajustar el ambiente para tener ese carácter como el carácter erase. Esto se hace usando el siguiente comando:

```
stty erase ^E
```

Aquí ^E es el carácter que se elige como carácter erase. Por defecto, el carácter proporcionado por el sistema es ^X.

Un solo carácter se puede usar para borrar una linea ingresada. Este se llama carácter ‘kill’. El carácter kill por defecto es ^U. Se puede cambiar el carácter kill a un carácter de su propia elección, tal como se muestra:

```
 /home/edwar$ sty kill ^K
/home/edwar$
```

Ahora ^K será el nuevo carácter kill personalizado. Considere este ejemplo:

```
/home/edwar$ echo this is my boo^K
/home/edwar$
```

Esto matara la linea donde se encuentre el carácter ^K.

El problema se presenta cuando se tiene que ingresar estos comandos cada vez que se conecta. Personalizar el ambiente de procesos es una forma bastante simple y poderosa de lograr un ambiente de trabajo personalizado. La mayoría de las personalizaciones viene del uso modificado de variables shell. Se estará tratando con personalización del ambiente de usuarios en la *Unidad 6 – Personalización del Ambiente de Usuario*.

7.2 Cambiar el Valor de PATH

Una de las variables shell mas importantes que controla donde el shell busca los comandos a ejecutar, es PATH. Por defecto, cuando se ingresa un comando, el shell busca el comando en el directorio actual, luego en /bin, seguido por /usr/bin/. /bin y /usr/bin que son directorios disponibles en todos los sistemas Linux que almacenan comandos.

Esta secuencia de directorios se denomina la *ruta de búsqueda (search path)* y se almacena en la variable PATH, para especificar la ruta de búsqueda.

Uso de la Variable Shell PATH

```
/home/edwar$ echo $PATH
        /opt/java/bin/:/opt/java/bin:/home/edwar/bin:/usr/local/bin:/usr/X11R6/bin:/usr/games
        :/opt/kde/bin:/gnome/bin:/usr/lib/mit/sbin:/usr/lib/qt3/bin:/opt/IBMJava2-
        142/bin:/usr/local/lib/xerces-c-src_2_7_0/bin:/usr/local/pgsql/bin
/home/edwar$
```

El símbolo : (dos puntos) separa dos directorios
Si se establece PATH a

```
/home/edwar$ PATH=. :/usr/bin:/bin:/home/edwar/book
/home/edwar$
```

El símbolo . (punto) representa el directorio actual. Se ha cambiado la secuencia de path. Se ha retirado /us/local/bin/ de path y agregado /home/edwar/book a path.

```
/home/edwar$ echo $PATH
. : /usr/bin:/bin:/home/edwar/book
/home/edwar$
```

También se puede agregar un nuevo conjunto de directorios a una ruta existente

```
/home/edwar$ PATH=PATH :/usr/local/bin
. : /usr/bin:/bin:/home/edwar/book:/usr/local/bin
/home/edwar$
```

Se agrego ahora /usr/local/bin a path

Fin de Uso de la Variable Shell PATH

7.3 Configurar el Tipo de Terminal

Los programas como vi requieren que el tipo de terminal se configure correctamente. La variable shell TERM se puede usar para este propósito. Algunos tipos de terminales son:

- ANSI
- Adm5
- Vt100

Si se usa TERM=vt100, se establece el tipo de terminal a vt100. Los tipos de terminales mas usados son ANSI y vt100.

También se puede manipular el ambiente de procesos creando abreviaciones (shortcuts) a directorios o nombre de archivos largos.

7.4 Crear Abreviaciones

Se puede desear adecuar el ambiente a que tenga shortcuts a abreviaciones para comandos, nombres de archivos y nombres de directorios. Por ejemplo, el nombre de directorio /home/edwar/cprogramas/projects/share se puede tener que usar repetidamente. Se puede configurar abreviado como se muestra a continuación:

```
/home/edwar$rep= /home/edwar/cprogramas/projects/share
/home/edwar$
```

Esto permite decir cd $rep para cambiar de directorio. Así, cuando se dice:

/home/edwar$ cd $rep
/home/edwar/cprogramas/projects/share$
Ahora, el directorio actual de trabajo, se vuelve el directorio share.
También se vera brevemente como las variables pueden ser exportados a un proceso hijo.

7.5 Exportar la Configuración Personalizada a un Proceso Shell Hijo

Se ha visto como personalizar algunas de las variables shell. Para asegurar que estas configuraciones estén presentes en todos los procesos shell hijo que el padre shell crea, se debe usar el comando export en Linux. Este comando toma la lista de variables shell que se desee exportar.
/home/edwar$ export PATH TERM
/home/edwar$
Si el comando export se usa sin ningún argumento, lista el conjunto de todas las variables shell que son exportadas a un proceso shell hijo. Se aprenderá acerca del proceso hijo y el comando export en detalle en la *Unidad 6 – Personalización del Ambiente de Usuario*.
Manejar un ambiente de procesos es importante porque un proceso puede acceder y usar cualquier de las variables del entorno. Dar la posibilidad al usuario de configurar las variables ofrecidas por el shell, es una flexibilidad permitida por el sistema Linux. Los usuarios pueden personalizar el ambiente para ajustarlo a sus necesidades.
A continuación se aprenderá acerca de los procesos demonios.

8. Proceso Demonio

Un programa que corre en segundo plano y atiende una solicitud legal, se denomina proceso demonio. Algunos procesos que corren como demonios son:

- Demonio de la cola de impresión (print spooler)
- Demonio del listener de red TCP
- Demonio del listener de correos

Cuando el sistema operativo se carga en la memoria principal, el kernel verifica si los discos son apropiados y si el sistema de archivos esta en su lugar. Luego ejecuta los demonios que estén programados para correr en segundo plano sin interrupción.
Los tipos de demonios que corren bajo un sistema, son específicos a ese sistema. Generalmente, un demonio de impresión esta presente en la mayoría de los sistemas. Cuando se emplea el comando ps –e se encuentran muchos procesos del sistema, algunos de los cuales se ejecutan como demonios. Algunos de los demonios que se ejecutan en el sistema Linux son:

lpd	Controla la cola de impresión para cualquier trabajo a ser impreso
getty	Monitorea el proceso de inicio de sesión de terminales
atd	Verifica cuando ejecutar los trabajos programados usando el comando at
crond	Verifica cuando ejecutar los trabajos programados en el proceso cron

Nota: los demonios no se ejecutan basados en la iniciación del usuario. No están asociados con un terminal. Así, la salida del comando ps –ef muestra un? para indicar que es un demonio.

9. Señales

En el contexto del comando kill usado para terminar un proceso en ejecución, se introdujo el término de señal. Cuando se presiona <ctrl + c>, se genera una señal. Esta es la tecla de interrupción que interrumpe y termina el comando que se esta ejecutando. El comando kill envía una señal a un proceso para que sea terminado.
Un proceso Linux puede recibir distintos tipos de señales. Algunos de los nombres simbólicos asociados con las señales se muestran en la Tabla 5.1

#Señal	Nombre simbólico	Descripción
1	SIGHUP	Suspensión
2	SIGINT	Interrupción
3	SIGQUIT	Salida como <ctrl + \>
4	SIGILL	Instrucción ilegal
8	SIGFPE	Excepción de punto flotante
9	SIGKILL	No puede ser ignorada
12	SIGSYS	La llamada al sistema tiene un argumento errado

Tabla 5.1 Señales en Linux

Linux proporciona un comando llamado trap, que permite que las señales de una aplicación sean capturadas. El comando trap toma un comando y una lista de señales.
La sintaxis de trap es como sigue:

trap comando lista-de-señales.

El comando espera que cualquiera de las señales mencionadas en la lista se lleve a cabo. Al recibir la señal, se ejecuta command. Las interrupciones a un proceso manejadas de esta forma, aseguran una salida limpia del proceso. Esto es una facilidad al correr aplicaciones críticas y complejas.

Unidad 6: Personalizar el Ambiente de Usuario

Objetivos de Aprendizaje

Al final de esta unida Ud. será capaz de:

- Personalizar las variables de entorno en una instalación Linux
- Explicar la funcionalidad de los archivos .bash_profile y .bashrc
- Describir un alias y sus usos
- Explicar el comando history

1. Configuraciones Personalizables

En la Unidad 3: *Características del Shell*, se indico que si se desea cambiar la cadena del prompt, se debe hacer cada vez que se ingresa al sistema. En esta unidad, se aprenderá como almacenar la cadena prompt requerida para que cada sesión de ingreso no requiera que el usuario explícitamente lo cambie, además de cómo hacer esto en la sección .bash_profile.

En la unidad anterior, se explico como se puede administrar el ambiente de procesos. Se aprendió a establecer los caracteres kill y erase y se explico como cambiar las variables shell PATH y TERM. En todos estos casos, se tiene que ingresar el comando cada vez que se conecta.

Se usara un ejemplo para entender esto. El usuario se conecta, realiza las siguientes tareas y finalmente se desconecta.

```
Login: edwar
Password: **********
edwar@localhost :~> PS1=$PWD$
/home/edwar$ echo PATH
/usr/local/bin:/usr/bin:/home/edwar/bin
/home/edwar$ PATH=$PATH:/usr/local/bin
/home/edwar$ echo $PATH
. :/usr/bin:/bin:/home/edwar$/book:/us/local/bin
/home/edwar$ exit
```

El usuario edwar cambio la cadena del prompt a /home/edwar$ y también cambio la variable PATH.

Al salir y conectarse nuevamente, el usuario encuentra lo siguiente:

```
Login: edwar$
Password: **********
edwar@localhost:~> echo PATH
/usr/local/bin:/usr/bin:/home/edwar$/bin
edwar@localhost:~>
```

Ambas variables shell establecidas por el usuario se han revertido a sus valores anteriores. Esto ocurre, porque estas configuraciones eran validas solo para la sesión de ingreso actual.

Al personalizar el ambiente se puede ir un paso adelante administrando justo el ambiente de procesos. Personalizar el ambiente de trabajo es una forma simple pero poderosa para lograr un ambiente de trabajo personalizado. La mayor parte de parte de la personalización se puede lograr modificando las variables shell.

En esta unidad, se aprenderá a personalizar lo siguiente:

- Configuraciones del Ambiente
- Scripts de Arranque del Shell.
- Alias
- Historial de Comandos

2. Configuraciones del Ambiente

Cada programa se ejecuta en un ambiente. El shell en el que el programa se esta ejecutando define ese entorno. El entorno existe dentro del shell. El entorno esta definido por los valores asignados a las variables de entorno. El comando export se usa para agregar o modificar variables de entorno. Se va a aprender como usar el comando export en las variables de entorno.

2.1 El comando export

Linux proporciona la capacidad de correr un shell dentro de otro. Generalmente, las variables declaradas en un shell son locales al shell y están disponibles solo dentro de ese shell. Si se corre otro shell dentro del shell actual y existe la necesidad de usar una de

las variables definidas en el shell padre (al shell que crea otro shell), entonces el shell padre debe exportar la variable shell al shell hijo. Simplemente invocando bash en el prompt se puede correr un nuevo shell.
Entiéndase esto con un ejemplo.

```
edwar@localhost:~> color=azul
edwar@localhost:~> echo $color
azul
edwar@localhost:~>
```

Se creo una variable shell color y se le asigno el valor azul. La variable color es local al shell actual. Ahora se va a crear otro shell desde el shell actual.

```
edwar@localhost:~> bash
edwar@localhost:~>
```

El comando anterior bash crea un shell dentro del shell actual. Se va intentar lo siguiente:

```
edwar@localhost:~> echo $color
edwar@localhost:~>
```

Se encuentra que el valor azul no se muestra ya que se esta en el shell hijo y el shell padre no ha exportado la variable color a sus shell hijos. El comando export debe usarse antes de crear un shell hijo. Ahora se va a realizar de la forma correcta. Para exportar la variable, se debe ingresar al shell padre. Se hace esto presionando <ctrl + d> o ingresando exit.

```
edwar@localhost:~> exit
edwar@localhost:~> export color
edwar@localhost:~> bash
edwar@localhost:~> echo color
azul
edwar@localhost:~>
```

Moviéndose al shell padre usando exit en la secuencia de pasos anterior, se invoca el comando export para exportar la variable shell color. Se invoca bash nuevamente para crear un shell hijo. Ahora el valor color es visible en el shell hijo. Las variables exportadas se denominan *variables globales*, ya que son visibles a otros shells creados dentro de un shell.
Así como con las variables shell, el símbolo $ se usa para evaluar variables de entorno. El uso del símbolo $ para evaluar la variable de entorno se hace solo en el contexto del shell, mientras el shell esta interpretando. El shell realiza la interpretación cuando los comandos son ingresados en el prompt o cuando bash lee los comandos de un archivo como .bashrc o .bash_profile. Estos dos archivos son importantes para un shell. Ahora se va a explicar el archivo .bash_profile.

2.2 Archivo .bash_profile

Hay un numero de archivos que empiezan con . (punto). Estos son archivos especiales y los usa el sistema para varios propósitos. .bash_profile es uno de esos archivos. Cada directorio home tiene una copia del archivo .bash_profile. Este archivo se usa para personalizar la configuración del usuario. .bash_profile se ejecuta solo una vez cuando el usuario se conecta al sistema. En cada sesión de ingreso, el usuario tendrá que ejecutar el archivo .bash_profile.
Asuma que la entrada en .bash_profile es como se muestra:

```
# .bash_profile
# Obtener alias y funciones
if [ -f  ~/.bashrc ] ; then
        . ~/.bashrc
fi
# Ambiente especifico del usuario y programas de arranque
PATH=$PATH:$HOME/bin
```

```
export PATH
unset USERNAME.
```

Se pueden agregar mas entradas al archivo simplemente invocando vi .bash_profile y agregando lineas. Se va a agregar algunas entradas, marcadas en negrita aquí.

```
# .bash_profile
#  Obtener alias y funciones
if [ -f  ~/.bashrc ] ; then
        . ~/.bashrc
fi
# Ambiente especifico del usuario y programas de arranque
PATH=$PATH:$HOME/bin
export PATH
unset USERNAME.
date
PS1=$PWD$
```

Una vez incluidas esas dos lineas, se guarda el archivo, se sale del sistema y nuevamente se vuelve a conectar. Esto es lo que se observa en la pantalla.

```
Welcome to SuSE Linux 10.3 (i586)
Kernel 2.6.4-52-default
Login: edwar
Password:**********
Last login: Wed Abr 08 14:28:40 on tty2
Wed Abr 08 14:35:26 IST 2009
/home/edwar$
```

Se observa que se muestra la fecha y hora actual y la nueva cadena del prompt. De esta forma, los usuarios pueden personalizar el ambiente a su preferencia, usando el archivo .bash_profile.

Una pregunta puede formularse. ¿Cómo se visualiza los archivos que empiezan con punto (.)?

Usando ls –al.

```
 /home/edwar$ ls –al
Total 60
 drwx------   4   edwar   lgrp   4096   Abr 30   11:20 .
drwxr-xr-x   10   root   root   4096   Abr 29   09:20 ..
-rw-------   1   edwar   lgrp   2624   Abr 30   11:20  .bash_history
-rw-r—r--   1   edwar   lgrp    24   Abr 30   09:50  .bash_logout
-rw-r—r--   1   edwar   lgrp    206   Abr 30   11:20  .bash_profile
-rw-r—r--   1   edwar   lgrp    124   Abr 30   11:20  .bashrc
/home/edwar$
```

2.3 El archivo .bashrc

Los archivos que tienen el sufijo rc tienen un significado especial. rc son las siglas de ‘run control’. Estos archivos permiten que los archivos sean configurados de acuerdo a lo que los usuarios desean. .bashrc se refiere al archivo de configuración especial para el programa bash. Cada directorio home tiene una copia del archivo .bashrc

En el archivo .bashrc, se pueden ubicar los comandos shell que se van a ejecutar cada vez que se inicia un nuevo programa. La entrada por defecto del archivo .bashrc se muestra a continuación:

3. El comando env

El comando env es muy utilizado para consultar el ambiente. Muestra todas las variables de entorno. Algunas de las variables de entorno se muestran en la Tabla 6.1

Nombre de la variable de Entorno	Descripción	ejemplo
EDITOR	Para establecer el editor por defecto. Generalmente se configura a emacs o vi	EDITOR=emacs EDITOR=vi
HOME	Para establecer el directorio de inicio (home) del usuario	HOME=/home/edwar
SHELL	Para establecer el programa shell actual en ejecución	SHELL=/bin/bash
TERM	Para establecer el tipo de terminal que se esta usando	TERM=ansi
USER	Para establecer el nombre del usuario actual	USER=edwar

Tabla 6.1: Muestra de las Variables de Entorno

Observe la salida del comando env

```
# .bashrc
# Obtener alias y funciones
# Definiciones de variables globales
if [ -f  ~/.bashrc ] ; then
        . ~/.bashrc
fi.
```

Se puede agregar comandos shell y establecer las variables shell que se desea ejecutar inmediatamente al ingresar. Se van a agregar las siguientes lineas al archivo .bashrc existente.

```
PS1=$PWD$
date
        Las nuevas entradas en .bashrc se muestran a continuación.
# .bashrc
# Obtener alias y funciones
# Definiciones de variables globales
if [ -f  ~/.bashrc ] ; then
        . ~/.bashrc
fi
whoami
```

Un punto muy importante a notar aquí, es que cada vez que se ejecuta la siguiente impresión en pantalla:

```
/home/edwar$ bash
edwar
/home/edwar$
/home/edwar$ bash
edwar
/home/edwar$
```

Si se desconecta y se conecta nuevamente, encontrara la siguiente impresión en pantalla:

```
Welcome to SuSE Linux 10.3 (i586)
Kernel 2.6.4-52-default
Login: edwar
Password:**********
Last login: Wed Abr 08 14:28:40 on tty2
Wed Abr 08 14:35:26 IST 2009
/home/edwar$
```

Se observa que se muestran las entradas adicionales hechas en .bash_profile y .bashrc. Si se hubiera mencionado también la fecha en .bash_rc, se hubiera observado que se

mostrarían dos salidas de fechas.
La principal diferencia entre .bash_profile y .bashrc es que mientras .bash_profile se ejecuta una sola vez cuando el usuario se conecta, .bashrc se ejecuta por cada ejecución de bash. .bashrc proporciona una forma por la cual se puede personalizar un programa, el programa bash en este caso.
Hay algunas variables que no son configuradas normalmente en el archivo .bashrc, ya que son importantes para una sesión de ingreso y no para un proceso shell. Algunas de estas HOME y USER.
De manera similar se tiene a .bash_logout, que se ejecuta cuando el usuario se desconecta del sistema.
A continuación se da a conocer el comando env.

```
/home/edwar$ env
PWD=/home/edwar
REMOTEHOST=192.168.1.201
HOSTNAME=localhost
PVM_RSH=/usr/lib/qt-2.3.1
LESSOPEN= |/usr/bin/lesspipe.sh %s
XPVM_ROOT=/usr/share/pvm3/xpvm
KEDIR=/usr
USER=edwar
LS_COLORS=
MACHTYPE=i686-suse-linux
MAIL=/var/spool/mail/edwar
INPUTRC=/etc/inputrc
LANG=en_US
LOGNAME=edwar
SHLVL=1
SHELL=/bin/bash
HOSTTYPE=i386
OSTYPE=linux
HISTSIZE=1000
LAHMELPFILE=/etc/lam/lam-helpfile
PVM_ROOT=/usr/share/pvm3
TERM=linux
HOME=/home/edwar
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
PATH=/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/edwar/bin
_=/usr/bin/env
/home/edwar$
```

Generalmente, los shell utilizan determinados shell scripts para realizar muchas de las tareas. .bash_profile es un ejemplo de un shell script. A continuación se dará una breve explicación acerca de los shell scripts.

4. Shell Scripts

Un bash puede ejecutarse como un login shell, un shell interactivo o un shell no interactivo. Un determinado conjunto de programas se ejecutan en secuencia cuando un shell arranca. Estos programas se denominan shell scripts o simplemente scripts. Se esta trabajando con Bourne Again Shell, y por lo tanto, se estará usando algunos scripts relacionados a bash. Algunos de scripts que se ejecutan comúnmente durante la sesión de ingreso y desconexión son .bash_profile, .bashrc y bash_logout. Estos scripts están disponibles en $HOME. El .bash_profile se ejecuta antes de un .bashrc.
Puede haber otros scripts que se ejecutan dependiendo de la versión e implementación especifica del sistema. Un archivo llamado .inputrc también puede ser ejecutado. Este

contiene alias y funciones de usuarios. Un archivo llamado profile en el directorio /etc se usa para modificar los parámetros de todos los usuarios en el sistema.
Otros archivos del directorio home del usuario pueden ser:
.emacs: Para el editor emacs
.exrc: Para el editor vi
.newsrc: Para newsreader (grupo de noticias). Esto es relevante cuando un conjunto de nuevos grupos de noticias se ofrece en la maquina.
Los usuarios pueden dar diferentes nombres a los comandos que utilizan frecuentemente. Esta facilidad se denomina alias, la cual se explicara a continuación.

5. Alias

El alias es un metodo por el cual el sistema Linux permite al usuario dar nombres cortos a los comandos. Observe el siguiente ejemplo:
/home/edwar$ alias dt=date
/home/edwar$
El comando date ha sido abreviado como dt. Ahora se puede usar dt para mostrar la fecha. Si se ingresa dt en el prompt de comandos, se obtiene la salida del comando date.
/home/edwar$ dt
Mie abr 08 16:46:08 VET 2009
Un punto importante a notar aquí es que las opciones para un comando se dan cuando se crea un alias. El alias si puede tomar una opcion. Si se desea abreviar ls –al, se puede hacer lo siguiente:
/home/edwar$ alias la=”ls –la”
/home/edwar$
Note que el comando con la opcion esta encerrado con comillas dobles. Si no se encierra, se mostrara el siguiente mensaje en pantalla:

bash: alias: `-al ’ not found

El shell trata ls y al como dos comandos diferentes.
También se pueden usar tuberías y filtros en un alias. Se va a presentar un ejemplo para entender esto. Asuma que se tiene un archivo llamado miarchivo.txt en el directorio home.
/home/edwar$ cat miarchivo.txt

This is all about aliasing
It is an interesting feature in Linux
I am enjoying learning this
Are there more such features?

/home/edwar$

Se van a crear algunos alias usando este archivo.

/home/edwar$ alias cf=”cat miarchivo.txt”
/home/edwar$ cf

This is all about aliasing
It is an interesting feature in Linux
I am enjoying learning this
Are there more such features?

/home/edwar$
cf es ahora el alias para el comando cat miarchivo.txt
/home/edwar$ alias pm=”cat miarchivo.txt | grep feature”
/home/edwar$ pm

It is an interesting feature in Linux
Are there more such features?

/home/edwar$.
La salida del comando cat se esta enviando a grep. Para esto se ha creado un alias llamado pm.
Se muestra otro ejemplo:

```
/home/edwar$ alias pmwc=”cat miarchivo.txt | grep feature | wc”
/home/edwar$ pmwc
        2       12      69
/home/edwar$
```

Estos alias pueden ser agregados a los archivos .bash_profile y .bashrc. agregar alias comunes a uno de estos archivos, asegurara que no se necesite volver a crear estos alias para cada sesión de ingreso.

6. Comando History

Cuando se trabaja en un sistema largo tiempo, se ejecutan numerosos comandos y se puede necesitar ejecutar algunos de los comandos anteriores. Así cada comando que se ingresa se almacena en un archivo llamado .bash_history. Este archivo esta disponible en el directorio home del usuario. Al desconectarse del sistema, los comandos usados se agregan al archivo .bash_history.

En cualquier momento se pueden encontrar los últimos n comandos utilizados en el sistema. El comando history seguido de un número, muestra los últimos n comandos utilizados. Observe el siguiente ejemplo:

```
/home/edwar$ history 10
        325     vi      .bash_profile
        326     vi      .bashrc
        327     vi      .bash_profile
        328     exit
        329     cf
        330     man history
        331     clear
        332     vi      .bash_history
        333     history         4
        334     history         10
/home/edwar$
```

En el ejemplo anterior, hasta ahora 334 comandos han sido agregados al archivo .bash_history. El comando history ha listado los últimos 10 comandos, que también incluye el comando actual que esta ejecutando, es decir history 10.

Ahora que se conoce el número del comando, se ejecuta el comando usando:

```
/home/edwar$ ! 329
cf
        This is all about aliasing
        It is an interesting feature in Linux
        I am enjoying learning this
        Are there more such features?
/home/edwar$
```

El símbolo ! se usa antes del numero para ejecutar el comando. El comando se muestra, seguido de la salida del comando. El numero 329 fue cf, el cual era un alias que se había creado.