TALLER GNU LINUX

Corporación Parque Tecnológico de Mérida
Javier Gutierrez
Gilberto Diaz

Índice

1. Introducción al Sistema Operativo GNU Linux **7**
- 1.1. Historia . . . 7
- 1.2. Distribuciones . . . 8
- 1.3. Características . . . 8
  - 1.3.1. Tiempo Compartido . . . 8
  - 1.3.2. Multitarea . . . 8
  - 1.3.3. Multiusuario . . . 8
  - 1.3.4. Núcleo Codificado en Lenguaje de Alto Nivel . . . 9
  - 1.3.5. Diseñado Originalmente para Programadores . . . 9
- 1.4. Componentes . . . 9
  - 1.4.1. Hardware . . . 9
  - 1.4.2. Núcleo . . . 9
  - 1.4.3. Conchas (Shells) . . . 9
  - 1.4.4. Programas de Aplicación . . . 10

2. El Ambiente del Usuario **10**
- 2.1. Entrada al sistema . . . 11
- 2.2. Interpretadores de comandos (Conchas) . . . 12
  - 2.2.1. Tipos de Interpretadores . . . 12
  - 2.2.2. Archivos de configuración de las Conchas . . . 13
  - 2.2.3. Características de las Conchas . . . 14
- 2.3. Ambientes de Ventanas . . . 16

3. Edición de Archivos **16**
- 3.1. Vi . . . 17
- 3.2. Otros editores (emacs, pico, joe). . . . 22

4. Comandos Básicos **22**
- 4.1. Manejo de archivos . . . 24
  - 4.1.1. Listado de archivos . . . 24
  - 4.1.2. Copiando Archivos . . . 25
  - 4.1.3. Moviendo archivos . . . 26
  - 4.1.4. Borrando Archivos . . . 26
  - 4.1.5. Visualizando el contenido de los archivos . . . 27
- 4.2. Manejo de directorios . . . 29
  - 4.2.1. Visualizando el directorio de trabajo . . . 29

4.2.2. Cambiando el directorio de trabajo 29
4.2.3. Creando nuevos directorios 29
4.2.4. Eliminando directorios 30
4.3. Búsqueda de archivos 30
4.4. Manejo de permisos 31
4.5. Manejo de Medios de Almacenamiento Secundario 34

5. Manejo de la Red (comandos de comunicación) 34
5.1. TELNET 34
5.2. FTP 35

6. Comandos Avanzados 37
6.1. Ordenamiento de Archivos 37
6.2. Búsqueda de Cadenas de Caracteres en Archivos 38
6.3. Cortar y Pegar Archivos 38
6.4. Comparación de Archivos 40
6.5. Comparación de Directorios 40

7. Manejo de Procesos 41
7.1. Estados de los procesos 41
7.2. Cómo activar un proceso 44
7.3. Manipulación de trabajos 44
7.4. Cómo cancelar un proceso 44

8. Apendice A: Programación en las conchas 46
8.1. Estructura de un script 46
8.2. Manipulación de variables 47
8.3. Lectura y Escritura 47
8.4. Manipulación de parámetros en los programas 48
8.5. Estructuras de Decisión 48
8.6. Estructuras de Repetición 49
8.6.1. Lazo while 49
8.6.2. Lazo for 50

Acerca de este manual

Este es un Manual de soporte para el curso **Linux para Usuarios Básico y Avanzado**, el cual es una introducción al manejo del Sistema Operativo Linux.

Audiencia

Este manual, al igual que el curso, está dirigido a personas que han tenido muy poca, o ninguna, experiecia con sistemas operativos compatibles con Linux. Intenta igualmente ser un tutorial organizado por tipo de tarea.

Objetivos

Este manual, en conjunto con el curso **Linux para Usuarios Básico y Avanzado** introduce al usuario a el uso de los comandos básicos y al uso de la concha `bash`. Al finalizar este manual usted debe estar en capacidad de:

- Acceder a su ambiente de trabajo en una máquina GNU Linux, ya sea por consola o desde la red.
- Entender y utilizar los comandos básicos del Linux
- Entender y utilizar los conceptos de directorios y archivos.
- Utilizar el editor de archivos `vi`
- Editar el archivo *.bashrc* para establecer comandos predeterminados
- Utilizar aplicaciones de red para comunicarse con otros sistemas para tranferencia de información o ejecución remota de procesos.

Organización

Este manual está organizado en 7 capitulos y 1 apéndice como sigue:

- **Introducción al Sistema Operativo Linux** Aquí se hace un recuento de la historia de este sistema operativo y se introducen algunos conceptos básicos y útiles para entender la relación de los sistemas operativos y las computadoras.

- **El ambiente del Usuario** Es una introducción a los ambientes de trabajo de los usuarios. Aquí se describen los interpretadores de comandos o *conchas* y los ambientes de ventanas de GNU Linux, en particular el ambiente **X**

- **Edicion de Archivos** Muestra el funcionamiento y comandos básicos del editor de archivos de Linux `vi`. Se hace, también, una introducción a algunos otros editores de archivos.

- **Comandos Básicos** Se hace una selección de los comandos de uso más frecuente o de importancia básica. Entre ellos los comandos de creación de directorios, copia de directorios o archivos, etc.

- bf Manejo de la red. Se describen dos de los protocolos más utilizados en internet: ssh, y FTP.

- **Comandos Avanzados** Se muestra el uso de comandos útiles pero de uso menos frecuente, en particular se presentan comandos para el manejo de archivos, comparación entre archivos o directorios y búsqueda de cadenas de caracteres.

- **Manejo de Procesos** Aquí se muestran los comandos de Linux que permiten tener control sobre el estado de los procesos que realiza una máquina.

- **Apendice A: Programación en las conchas** Se hace una introducción a los conceptos necesarios para controlar el ambiente de las *conchas* y para crear archivos ejecutables basados en comandos de Linux.

Información relacionada en el Web

En el web existen los siguientes documentos:

- A Basic UNIX Tutorial:
  http://www.isu.edu/departments/comcom/unix/workshop/unixindex.html

- Linux Online Courses
  http://www.linux.org/lessons/

- Introduction To UNIX:
  http://www.ceas.rochester.edu:8080/CNG/docs/IntroUnix.html

- Unix
  http://metalab.unc.edu/echernof/unix/what.html

Convenciones

En este manual se utilizan las siguientes convenciones:

`%`	Un signo de porcentaje al iniciar una linea en los ejemplos representa el *prompt* (mensaje de espera) de una concha tipo **C shell**
`$`	Un signo de dollar representa el *prompt* de una concha tipo **Bourne Shell**.
`#`	Un signo de número representa el *prompt* de superusuario.
`% cat`	La parte en **negrita** representa el comando a ser escrito por el usuario
archivo	Las partes en *itálicas* muestran las partes variables del comando que deben ser substituidas por el usuario para su caso especial.
`[ | ]`	En la definición de la sintáxis de un comando, los corchetes indican las partes opcionales y las barras verticales separan los parámetros entre si, indicando que debe ser introducido uno u otro.
`{ | }`	En la definición de la sintáxis de un comando, las llaves indican los parámetros requeridos y las barras verticales separan los parámetros entre si, indicando que debe ser introducido uno u otro.
`cat(1)`	Cuando los comandos aparecen de esta forma en el texto o en los ejemplos, se está haciendo referencia a la información obtenida a traves del comando **man**
`ctrl-x`	La secuencia de caracteres `ctrl-` delante de una letra indica que se debe mantener presionada la tecla **Ctrl** al mismo tiempo que se presiona la letra indicada.

1. Introducción al Sistema Operativo GNU Linux

1.1. Historia

" Hello everybody out there using minix - I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones..."
Linus Torvalds

¿Qué es GNU Linux? GNU Linux es un sistema operativo que controla los recursos de una computadora y la interacción del usuario con esta. Permite a uno o varios usuarios correr sus programas y controla los dispositivos externos, como cintas, impresoras, terminales, etc. Por la forma como maneja los recursos de la máquina y la política de tiempo compartido entre procesos, GNU Linux es un sistema operativo multiusuario.

Al inicio de la década de los noventa el sistema operativo predominante en el mundo de los computadores personales era MSDOS (Microsof Disk Operating System). Las computadoras Apple constituian una alternativa poco accesible debido a sus precios elevados.

Otro de los sistemas operativos importantes en esa época era Unix, desarrollado en los laboratorios Bell durante 1969 por Ken Thompson, Rudd Canaday, Doug McIlroy, Joe Ossanna y Dennis Ritchie. El año 1974 se introdujo Unix en alguna Universidades con "fines académicos" y al cabo de pocos años ya estaba disponible comercialmente. Sin embargo, los proveedores mantuvieron los precios elevados de tal manera que Unix se mantuvo lejos del ambiente de los PCs. La solución parecía ser MINIX, desarrollado por el profesor Andrew Tanenbaum para la enseñanza a sus estudiantes. Este sistema operativo no fue exitoso como tal pero existía la ventaja de que su código fuente estaba disponible al público.

GNU Linux es un *clon* del sistema operativo Unix, inspirado en MINIX, de libre distribución, diseñado originalmente para máquinas 80386 y 80486. El sistema operativo GNU Linux fue desarrollado por Linus Torvalds en la Universidad de Helsinki - Finlandia en 1991. La versión 0.01 fue liberada por Linus en Septiembre de ese año.

1.2. Distribuciones

El movimiento GNU ha generado una gran cantidad de aplicaciones y programas que realizan diferentes tareas y están dedicadas a diferentes propósitos. Organizaciones como Red Hat, Mandrake, SUSE, Debian, Gentoo, etc, recopilan su propio conjunto de esas aplicaciones, modifican el núcleo y elaboran sus propios programas de instalación. Esta compilación de aplicaciones, núcleo e instaladores conforman lo que se conoce como **distribución**.

1.3. Características

El sistema opetativo GNU Linux presenta una serie de características que lo hace atractivo a los usuarios y adecuado para desempeñarse en la mayoria de las áreas importantes de la computación. Está ampliamente difundido en máquinas que fungen como servidoras de base de datos, procesamiento de información, almacenamiento, etc.

1.3.1. Tiempo Compartido

Esta característica es la base de mucha de las propiedades de GNU Linux. El tiempo compartido consiste en una cola de procesos listos para entrar al procesador y ser ejecutados. A cada proceso se le asigna un lapso de tiempo determinado, denominado **quantum**, para permanecer dentro del procesador. El proceso abandona el procesador cuando finaliza o cuando su quantum ha expirado. En este último caso el proceso regresa a la cola para esperar de nuevo por el procesador.

1.3.2. Multitarea

Gracias a la existencia de una cola de procesos listos, se puede cargar más de una tarea en memoria, y entrar a esta cola para competir por el procesador. La ejecución de procesos se hace de forma secuencial, sin embargo, la rapidez con que se mueve el flujo de procesos en el sistema crea la ilusión de ejecución concurrente de trabajos.

1.3.3. Multiusuario

GNU Linux es capaz de distinguir entre diferentes usuarios que entran y utilizan el sistema. Esto significa que dos o más personas pueden ejecutar

tareas simultáneamente en el mismo procesador. Para establecer la diferencia entre los distintos usuarios se le asigna un nombre, **login**, a cada uno de ellos.

1.3.4. Núcleo Codificado en Lenguaje de Alto Nivel

Linus Torvalds escribió el núcleo de GNU Linux en lenguaje C. Por esta razón es muy fácil trasladar el sistema operativo a cualquier plataforma.

1.3.5. Diseñado Originalmente para Programadores

Originalmente GNU Linux estaba dirigido a un tipo de usuarios particular, programadores. Por lo tanto, cuenta con una serie de herramientas adecuadas para el desarrollo de programas. Actualmente, GNU Linux está capacitado para dar soporte a un número considerable de áreas, como por ejemplo, bases de datos, servicios de red, control de sistemas, comunicaciones, visualización, cálculo intensivo, etc.

1.4. Componentes

1.4.1. Hardware

GNU Linux ha sido trasladado a la gran mayoría de las plataformas actuales, desde las computadoras de mano (Pam top) hasta las máquinas más grandes del planeta (supercomputadoras). Esto ha sido posible gracias a que su núcleo ha sido codificado en lenguaje de alto nivel.

1.4.2. Núcleo

El núcleo conforma el corazón del sistema operativo. Este es un procesos que siempre se encuentra en memoria principal y es el encargado de controlar el hardware, administrar los procesos, asignar memoria a los procesos, gestionar los dispositivos periféricos de la máquina, etc.

1.4.3. Conchas (Shells)

Los interpretadores de comandos, o conchas, conforman la interfaz mediante la cual los usuarios pueden comunicarse con el sistema operativo y ejecutar sus requerimientos. Estos son programas comunes que se encargan de leer, interpretar y ejecutar las intrucciones dadas por los usuarios. En

la actuliadad, existen diferentes tipos de conchas, cada una con sus propias características y ventajas.

1.4.4. Programas de Aplicación

GNU Linux actualmente incorpora de una serie de aplicaciones especializadas para ejecutar diversas tareas que van desde el procesamiento de imágenes, pasando por el procesamiento de textos, bases de datos, etc, hasta juegos. Una instalación estándar de GNU Linux puede contar con más de dos mil comandos de propósito general.

La figura 1 muestra la jerarquía de los componentes de un Sistema GNU Linux.

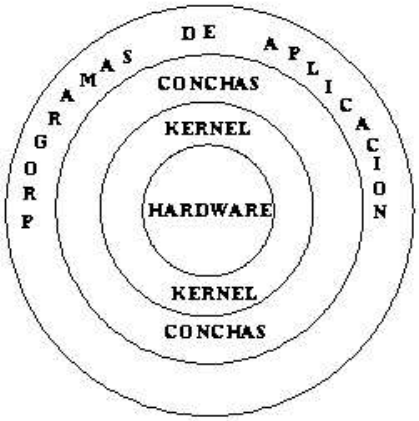

Figura 1: Componentes del Sistema Operativo GNU Linux

2. El Ambiente del Usuario

GNU Linux es un sistema multiusuario, por esto, es necesario establecer una configuración personal y un sistema de permisología para cada usuario. Esto permite que el usuario mantenga información confidencial en el sistema y que otro usuario no pueda accederla. Además, cada usuario puede tener una configuración para realizar unas u otras tareas sin afectar el trabajo de otros usuarios. Esta configuración, más la información de la estructura de

directorios que pertenecen al usuario, su *login* y su *password*, conforman lo que se llama una **cuenta de usuario**.

Las cuentas de usuario tienen varias utilidades, aún cuando la principal es la de permitir a un usuario entrar en el sistema y correr programas, existen cuentas especiales con privilegios y funciones particulares. La más importante de las cuentas especiales es la del administrador del sistema, esta es llamada *root* y tiene los máximos privilegios, es decir es la única cuenta que puede cambiar la configuración del sistema, instalar programas para todos los usuarios, apagar o reiniciar el sistema y tiene ademas el privilegio de leer todos los directorios del sistema, incluyendo los de los usuarios. Existen otras cuentas especiales como por ejemplo: *nobody* la cual no posee nigún privilegio y es utilizada para iniciar servicios de red que necesitan de cierto nivel de seguridad como lo es el servidor de web, etc. Nobody solo tiene acceso a su propio directorio y algunos comandos muy básicos. Otra cuenta especial es *lp*, en algunos sistemas se designa este usuario virtual para que controle los procesos de impresión.

2.1. Entrada al sistema

Una de las características de GNU Linux, al igual que todos los sistemas operativos modernos, es que GNU Linux es *full duplex*, es decir, que al escribir una letra esta aparecera reflejada de inmediato en el monitor. Sin embargo en algunos casos especiales este reflejo es cortado para seguridad del usuario o para asignar a la tecla una función diferente a la normal. Uno de estos casos es cuando se introduce el *password*, que el usuario escribe una serie de caracteres y en el monitor no aparece nada.

La unica forma de tener acceso a un sistema GNU Linux es a través de una cuenta de usuario. Se debe conocer el *login* o nombre de la cuenta y el *password* o clave de acceso.

Luego de introducir el *login* y el *password* ocurrirá un corto período de tiempo de espera, durante el cual el sistema valida la información suministrada, y una vez hecho esto aparecerá en pantalla algo como lo siguiente:

```
This is odie.ing.ula.ve (Linux i686 2.6.11.11) 16:28:41

odie login: gilberto
Password:
Last login: Fri Jul 29 16:33:11 on vc/1
```

```
gilberto en odie>
```

Esta información muestra generalmente la versión del Sistema Operativo GNU Linux que tiene el sistema y alguna otra información adicional que puede variar de acuerdo al sistema que se este utilizando.

Al ingresar el *login* y el *password* es importante recordar que GNU Linux hace diferencia entre letras mayúsculas y minúsculas, por ejemplo, si el password fue establecido como `Un_3T28aa`, debe ser escrito conservando el orden de letras mayúsculas y minúsculas.

2.2. Interpretadores de comandos (Conchas)

Las conchas o *shells* son los programas de GNU Linux que interpretan los comandos suministrados por el usuario; estas se presentan como una interfaz interactiva basada en texto. La primera concha UNIX, llamada *sh*, era una concha que ofrecía pocas posibilidades de interacción. Con el tiempo se fueron desarrollando conchas más amigables como la *csh*. Actualmente se pueden agrupar de la siguiente manera: dos conchas que utilizan una sintaxis similar a las *csh* (*csh* y *tcsh*) y cuatro que utilizan una sintaxis igual a la de *sh* (*sh*, *ksh*, *bash* y *zsh*). La última generación de conchas (*tcsh*, *ksh*, *bash* y *zsh*) introducen nuevas características como la de editar los comando en línea, la posibilidad de utilizar barras de desplazamiento (*scroll bar*), mecanismos para recuperar comandos anteriores (historia) y comandos para completar nombres (*command/file-name*).

Los *shells* modernos se han convertido en más que un simple interpretador de comandos. Estos *shells* poseen lenguajes de programación con posibilidades de utilizar estructuras elaboradas de decisión y de repetición. Esto permite la elaboración de rutinas o *scripts* basadas en comandos de GNU Linux y las estructuras del *shell* en uso y correrlos como nuevos comandos. Al correr una rutina escrita en el "lenguaje" *shell* se genera una nueva instancia de la concha, esta *subshell* corre el programa y al salir deja el *shell* padre intacto.

A través de los *scripts* se pueden realizar tareas tediosas y habituales con un solo comando.

2.2.1. Tipos de Interpretadores

Básicamente existen dos vertientes de interpretadores de comandos, los *C shell* y los *Bourne Shell*. La diferencia entre ambos es el estilo que se

utiliza para las funciones avanzadas como los son la definición de variables de ambiente, los *scripts* y la sintaxis en el lenguaje.

Como se mencionó anteriormente, el *prompt* por omision, que aparece en pantalla depende del tipo de concha que se utilice. Si tenemos una concha del tipo csh, el prompt será %, para las conchas sh o ksh tendremos $, el prompt # está reservado para el administrador del sistema o root. Si aparece un prompt más personal, como por ejemplo el nombre de la máquina, es porque en alguno de los archivos de configuración del usuario hay un comando que permite ponerle algún nombre al aviso de espera.

2.2.2. Archivos de configuración de las Conchas

Los archivos de configuración son básicamente archivos de comandos de GNU Linux que son leídos al iniciar una sesión en uno de los *shell*. En estos archivos se define el ambiente del usuario, que consta de la información sobre los caminos en los que busca los comandos, las variables de ambiente etc.

Para las conchas tipo *C Shell* existen dos archivos de configuración el *.login* y el *.cshrc*, mientras que en las conchas tipo *Bourne Shell* las configuraciones se hacen en los archivos *.profile* y *.kshrc* (si está utilizando **ksh**). El archivo de configuración para bash es .bashrc

Ejemplo de *.bashrc*:

```
# .bashrc

# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

MACHINE=‘hostname -s‘
USER=‘whoami‘

#PS1="$USER en $MACHINE>"
PS1="\[\033[01;34m\]$USER en \h\[\033[00m\]>"
PATH=/usr/local/mpich/bin:$PATH:/sbin:/usr/sbin:\
~/comandos:/usr/local/netscape:/
alias ls=’ls --color’
alias h=’history’
alias cs=’clear’
```

```
export HISTFILESIZE=5000
export HISTSIZE=5000
export TERM=xterm
```

El significado de cada línea será aclarado al transcurrir el manual.

2.2.3. Características de las Conchas

Las conchas poseen funcionalidades adicionales a la interpretación de comandos. Dentro de esas capacidades se tiene:

- **Ejecución de programas y secuenciamiento de comandos:** Cuando un usuario escribe los comandos, lo hace desde la línea de comandos. En general, ésta está conformada por un comando y sus argumentos. Esta línea es analizada por la concha, la cual es responsable de identificar el comando y de revisar si hay metacaracteres para realizar un procesamiento adicional sobre esos metacaracteres. Luego la concha arranca la ejecución del programa. Cuando esta ejecución termina el control vuelve a la concha. Por otra parte se puede ordenar la ejecución de varios comandos en secuencia en una misma línea de comandos, utilizando el caracter punto y coma ";".

  Ejemplo:

  % **ls;date**

  ```
  PROD3.txt           indice.aux  indice.dvi  indice.tex  prueba.ps
  PROD6.txt           indice.bbl  indice.log  indice.toc  tallerunix
  Sat Nov  6 03:01:23 VET 1999
  %
  ```

  Observe que después del listado de los archivos la fecha actual es mostrada, la cual es la salida del segundo comando.

- **Sustitución de nombres de archivos:** La especificación del nombre de un archivo puede ser generalizada a través del uso de caracteres tales como "*", "?". La concha se encarga de realizar la sustitución de tales caracteres en los nombres.

  Ejemplo:

  % **ls -l archivo?**

```
archivo1 archivo2 archivo3 archivoa archivoz
%
```

En este caso se indica que para el comando ls serán considerados todos aquellos archivos que empiecen con la palabra archivo y terminen con cualquier otro caracter, tales como archivo1, archivo2, ... archivoa, archivoz, etc.

- **Redirección de entrada/salida:** GNU Linux realiza el tratamiento de todos los componentes de una máquina mediante archivos especiales. Toda concha tiene asignado un dispositivo específico o archivo especial para la salida estándar, la entrada estándar y el despliegue de errores. Las conchas tienen la capacidad de utilizar archivos alternos para manejar la entrada, la salida y el despliegue de errores. Esto se logra a través del redireccionamiento, y para ello se utilizan los caracteres: $>, >>, <, <<, 2>, 2>>$.

  Por ejemplo, existe un archivo especial que maneja la pantalla el cual es designado como la salida estándar y también para el despliegue de errores. Cualquier comando ejecutado enviará su salida a tal archivo. Si se desea guardar esta salida en un archivo convencional, entonces deberá ser ejecutado el comando como sigue:

  % **comando** > **arch**

  Si el archivo arch no existe, será creado y contendrá la salida del comando. Si arch ya existia, entonces será sobreescrito y tendrá como contenido la salida del comando. Si el archivo ya existe y se desea conservar el contenido, se debería ejecutar el siguiente comando:

  % **comando** >> **arch**

  Esto adiciona la salida del comando al final del archivo.

- **Encauzamiento o pipes:** Otra capacidad de las conchas es poder redirigir la salida de un comando hacia otro comando. Este último, tomará como entrada la salida del primero. Para lograr esto, se utiliza el caracter "|" de la forma siguiente:

  % $\mathbf{cmd}_1$ | $\mathbf{cmd}_2$ |...| $\mathbf{cmd}_n$

- **Control del ambiente:** La concha permite adaptar el ambiente a las necesidades del usuario, a través de las variables de ambiente tales

como: PATH y HOME y además permite modificar el caracter de espera del sistema, prompt (El ambiente puede estar caracterizado por otras variables, manejadas también por la concha).

- **Interpretador de lenguaje de programación:** Las conchas "entienden" un lenguaje de programación. Un código en ese lenguaje puede ser escrito en la línea de comandos o guardado en un archivo para ser ejecutado posteriormente. En las próximas secciones se explicará el lenguaje de comandos de la concha Bash.

2.3. Ambientes de Ventanas

Existen diferentes manejadores de ventanas entre los sistemas GNU Linux. En la actualidad existen una gran variedad de ambientes de ventanas del dominio público como: kde, fvwm, afterstep, enlightenment, etc. los cuales pueden obtenerse de cualquier sitio en internet y ser compilado e instalado en cualquier plataforma GNU Linux.

Todos los ambientes de ventanas conocidos se apoyan en el sistema X Windows. El sistema X Windows es un estándar introducido por el MIT al mercado del UNIX, que encontró rápido apoyo en compañías como: IBM, SUN, DEC, HP y AT&T. En estos momentos la versión más utilizada y que se instala por omisión en casi todas las plataformas de GNU Linux es la versión 11 o X11. De esta versión se han construido varias reediciones conocidas como X11R5 y X11R6 (las más utilizadas).

El sistema X11 está basado en el modelo cliente-servidor para el despliegue gráfico. En el sistema corre un programa que controla la tarjeta gráfica y que es el servidor gráfico. Los programas que requieren una salida gráfica son los clientes y deben comunicarse con el servidor. Este modelo permite que el programa cliente se ejecute en una máquina remota en la red y el despliegue gráfico se haga en la máquina local.

3. Edición de Archivos

La edición de textos se realiza en la elaboración de un memo, un correo electrónico, en la modificación o creación de un codigo en C o Fortran, cuando se escribe un reporte o tesis. Por todo esto, la edición de textos es la tarea más común en una computadora. Para tal fin, al igual que otros sistemas

operativos, GNU Linux incluye un editor de archivos —el **ed**—, el cual es un editor de líneas poco ágil y engorroso. Por esto, en la mayoría de las distribuciones de GNU Linux se incluye uno o varios editores de texto además del **ed**. El editor de texto más difundido en el ambiente GNU Linux es el **vi**, el cual es un editor que trabaja en ambiente de texto, pero que permite (con la sola interacción del teclado) todas las funciones de los editores de texto mas modernos. Tambien se incluyen algunos editores de texto que funcionan en ambiente gráfico como el xedit, el nedit, etc.

Otro editor que es incluido en muchas distribuciones, sobre todo en el Linux, es el Emacs o su versión gráfica el XEmacs. Este se caracteriza por tener una gran biblioteca de macros que se configuran automáticamente dependiendo del tipo de archivo a editar, permitiendo al usuario resaltar de forma muy sencilla la sintaxis de los lenguajes de programación o de los programas de procesamiento de texto.

En esta sección se hara una introducción al **vi** debido a que es un estándar y no requiere la presencia de un manejador de ventana, lo que permite hacer edición remota de textos, de una forma eficiente. Si bien es cierto que el **vi** puede realizar todas las labores de un editor de textos modernos, su interfaz tipo texto hace un poco laborioso su aprendizaje. Sobre los editores adaptados a los ambientes gráficos sólo se hará un comentario debido a que los ambientes de ventana salen de la cobertura de este curso.

3.1. Vi

El **vi** es, hoy en día, el editor de texto por omisión del GNU Linux, y es llamado así por acrónimo de *visually oriented* (la palabra visually se debe a que antes de **vi** sólo existían editores de líneas y teletipos).

El **vi** tiene una serie de características que lo hacen el preferido de las personas que trabajan a menudo con máquinas conectadas en red, ya sea para edición remota de archivos de entrada o cambios menores en códigos fuentes, como para el trabajo local en edición de archivos más amplios como papers, tesis, etc. Estas características son:

- Procesamiento rápido, especialmente en el arranque y en las operaciones globales. Esto debido a que no tiene el peso de la interfaz gráfica.
- Edición en pantalla completa.

- Modos separados para edición o inserción de texto. Esto se hace nesesario por la imposibilidad de utilizar menus en modo texto, sin embargo hace que la edición de texto sea más segura, ya que no se puede insertar texto mientras se hacen las busquedas y otras operaciones globales.

- Sustitución global y ediciones complejas basadas en comandos `ex`, es cual es la base para toda una familia de editores de archivos en GNU Linux.

- Acceso a comandos del sistema operativo, lo que permite probar la sintaxis de código fuente, entre otros, sin necesidad de salir del **vi**.

- Habilidad de asignar macros a teclas y de personalizar el sistema.

- Posibilidad de efectuar comandos de edición a sectores del archivo, selecionados por número de línea.

Al editar un archivo en **vi** se tienen dos modos: el modo de comandos y el modo de inserción. El modo de comandos es el modo por omisión (al contrario que en la mayoría de los editores de texto para PC.). En este modo se puede navegar por el texto, borrar caracteres, borrar líneas, marcar, mover o borrar bloques, etc. En el modo de comandos se puede tener acceso a los comandos de `ex` comenzando por presionar la tecla ":".

Para cambiarse al modo de inserción se utilizan las letras `i`, `o`, `a`, `I`, `O`, `A`, `cw` y `CW`, dependiendo de la acción que se quiera realizar. Para volver al modo de edición de texto se utiliza la tecla `esc`. Esta tecla también se puede utilizar para cancelar un comando de `ex`.

Los comandos básicos en vi se dividen en varios grupos dependiendo de la función y de la forma de utilización. Los grupos principales son: los comandos de navegación, modificación, comandos de control o subcomandos, búsqueda y de copiado y pegado de texto. Cada grupo se describe brevemente en las tablas 1, 2, 3, 4, 5, respectivamente.

Cuadro 1: Comandos de Navegación

comando	**descripción**
flechas o hjkl	mueve el cursor un caracter.
w	hacia adelante una palabra.
b	hacia atrás una palabra.
e	hacia el final de la palabra actual.
ctrl-u	desplaza el texto hacia abajo media pantalla
ctrl-d	desplaza el texto hacia arriba media pantalla
ctrl-f	desplaza el texto hacia abajo una pantalla
ctrl-b	desplaza el texto hacia arriba una pantalla
0	va al principio de la línea
$	va al final de la línea
:1	va al principio del archivo
:$	va al final del archivo
G	va al final del archivo
n G	a la línea número n
:n	a la línea número n

Cuadro 2: Comandos de Modificación

comando	**descripción**
i	inserta texto antes del cursor
I	inserta al principio de la línea
a	añade texto después del cursor
A	añade texto al final de la línea
o	abre una nueva línea debajo del cursor
O	abre una nueva línea encima del cursor
x	borra un caracter (delete)
X	borra un caracter (backspace)
dw	borra una palabra
dd	borra una línea
r	reemplaza un caracter
s	sustituye un caracter por una cadena
S	sustituye la línea
cw	sustituye la palabra actual
c# w	sustituye las # palabras consecutivas
C	sustituye el resto de la línea
cc	sustituye la línea
u	deshace el último cambio
U	deshace todos los cambios en la línea actual
J	une líneas
Esc	termina un comando de inserción o reemplazo

Cuadro 3: Subcomandos

comando	**descripción**
:w	guarda el archivo
ZZ	guarda los cambios y sale del editor
:q!	sale sin guardar los cambios
:e!	edita de nuevo el archivo anterior, es decir, se devuelve a la última version guardada
:n	edita el próximo archivo, si el editor fue invocado con vi arch1 arch2 arch...
:f	muestra el archivo actual
:set nu	enumera las líneas

Cuadro 4: Comandos de Búsqueda

comando	**descripción**
/texto[enter]	busca el texto "texto" hacia adelante
?texto[enter]	busca el texto "texto" hacia atrás
n	busca la siguiente ocurrencia del texto
N	busca la ocurrencia anterior del texto

Cuadro 5: Copiar y Pegar

comando	descripción
Y o yy	YANK (copia) la línea actual
yw	YANK (copia) la palabra siguiente
p	pega la última copia antes del cursor
P	pega la última copia después del cursor
ma	marca el un extremo de un bloque de líneas
y'a	marca el otro extremo de un bloque de líneas y lo copia

3.2. Otros editores (emacs, pico, joe).

xedit

Es un editor básico que funciona en ambiente X, presenta la ventaja (sobre el vi estándar) de que permite que el usuario utilice el ratón para navegar por el texto, sin embargo tiene muy pocas opciones de edición, de hecho, sólo incluye macros para búsqueda y reemplazo de cadenas de caracteres.

emacs y xemacs

Es el editor de la Fundación de Software Gratis (GNU). Es personalizable, extensible, permite interactuar con el ratón y tiene ayuda en línea. El emacs, al contrario del vi está siempre en modo de inserción y los comandos no hacen diferencia entre mayúsculas y minúsculas. Una buena manera de empezar es por ejemplo:

```
yemanya% emacs [enter]
crtl-h t
```

de este modo se invoca el tutorial, que se puede seguir paso a paso para familiarizarse con la interfaz del emacs.

4. Comandos Básicos

GNU Linux tiene dos tipos de comandos, los comandos que forman parte del *shell* y los comandos del sistema. Es por esto que de aqui en adelante haremos uso de las conchas tipo *bash (Bourne again shell)*, ya que los comandos de las *C Shell* son diferentes los cuales se pueden encontrar con el

comando **man** (que estudiaremos más adelante), en las páginas `ksh(1)` y `sh(1)`.

Los comandos que forman parte de las conchas *Bash* se muestran en la tabla 6.

comando	**descripción**
`alias`	asigna y muestra una definición de alias
`bg`	Coloca un trabajo suspendido en ejecución de fondo
`echo`	Escribe el argumanto a la salida estandar.
`fg`	Pasa un trabajo, que esté en ejecución de fondo, a ejecución interactiva
`history`	Muestra el contenido de la historia de comandos
`jobs`	Muestra el numero de trabajo y el PID de los trabajos que están corriendo en el fondo
`logout`	termina la sesión de trabajo
`rehash`	Le indica al *shell* que debe recalcular la tabla de *hash*, de modo que pueda encontrar un comando recién instalado
`repeat`	Repite un comando un número específico de veces
`set`	Establece y muestra una variable de la concha
`env`	Establece y muestra una variable de ambiente
`source`	Ejecuta los comandos escritos en un archivo. Puede ser utilizado para actualizar el ambiente de la concha
`time`	Muestra el tiempo de ejecución de un comando
`unalias`	Elimina una definición de un alias
`unset`	Elimina una variable de la concha
`unsetenv`	Elimina una variable de ambiente

Cuadro 6: Comandos de las conchas tipos Bash

El segundo tipo de comando está conformado por una serie de programas cuyo comportamiento es independiente del tipo de concha, esto hace que se incremente la flexibilidad del sistema operativo, pues cada comando es un programa independiente con opciones y modificaciones. Estos comandos se explican en las siguientes secciones.

Para obtener información acerca del uso de estos comandos, GNU Linux cuenta con un manual en línea, que puede ser consultado a través del comando **man** como se muestra a continuación:

man [-k] [comando—palabra clave]

Por ejemplo:

man cp

```
CP(1)                                                          CP(1)

NAME
     cp, ln, mv - copy, link or move files

SYNOPSIS
     cp [ -firRp ] file1 [file2 ...] target
     ln [ -sif ] file1 [file2 ...] target
     mv [ -if ] file1 [file2 ...] target

DESCRIPTION

     file1 is copied (linked, moved) to target.  Under no
     circumstance can file1 and target be the same (take care
     when using sh(1) metacharacters). If target is a directory,
     then one or more files are copied (linked, moved) to that
     directory.  If target is an existing file, its contents are
     destroyed, except in the ln and ln -s case where the command
     will fail and ln will write a diagnostic message to standard
     error (use the -i or -f option to override this behavior).
     NOTE that this is a change from the historical ln execution.
     ....
```

4.1. Manejo de archivos

4.1.1. Listado de archivos

Para mostrar el listado de los archivos del directorio actual se utiliza el comando **ls**, el cual tiene la siguiente sintaxis:

ls [-RadLCxmlnogrtucpFbqisf1AM] [nombres]

Las opciones en el primer corchete pueden ser utilizadas solas o una combinación de ellas. Algunas de las opciones mas utilizadas son:

- **ls**
  Muestra una lista de los archivos del directorio.
- **ls -a**
  Despliega una lista de los archivos pero además muestra los archivos de

configuración que suelen llamarse “archivos escondidos” cuyos nombres comienzan con el caracter punto (.).

- **ls -l**
  Despliega una lista detallada de los archivos y directorios. Muestra los permisos, el número de enlaces, propietario, tamaño en bytes y cuando ocurrió la última modificación para cada uno de los archivos.

- **ls -F**
  Muestra una lista de archivos agregando una diagonal (/) al final de los nombres de directorio; un asterisco (*) si se trata de un archivo ejecutable; un arroba (@) si el archivo es un enlace simbolico y un igual (=) si el archivo es un socket.

Ejemplo:

% **ls**

```
Miscellaneous  default.gif    hec.html.bak   index.html     index.shtml
cgi-bin        hec.html       hec01.html     index.html.N
```

% **ls -a**

```
.              cgi-bin        hec.html.bak   index.html.N
..             default.gif    hec01.html     index.shtml
Miscellaneous  hec.html       index.html
```

% **ls -l**

```
total 64
drwxr-sr-x   2 hector   ciencias     512 Apr 19 17:50 Miscellaneous
drwxr-sr-x   2 hector   ciencias     512 Apr 19 17:50 cgi-bin
-rw-r--r--   1 hector   ciencias    2085 Dec 12 1996  default.gif
-rw-------   1 hector   ciencias    2645 Feb 27 1997  hec.html
-rw-------   1 hector   ciencias    1787 Feb 27 1997  hec.html.bak
-rw-------   1 hector   ciencias    1368 Feb 27 1997  hec01.html
-rw-r--r--   1 hector   ciencias     860 Dec 12 1996  index.html
-rw-r--r--   1 hector   ciencias     798 Oct  9 1997  index.html.N
lrwxrwxrwx   1 root     ciencias      10 Apr 25 15:44 index.shtml -> index.html
```

4.1.2. Copiando Archivos

El comando para copiar archivos o directorios tiene la siguiente sintaxis:

- **cp archivo1 archivo2** Copia el contenido del archivo archivo1 en el archivo archivo2.
- **cp arch1 arch2 .... dir1** Cada archivo de la lista será copiado en el directorio dir1. El directorio dir1 debe estar creado con anterioridad.
- **cp -r dir1 dir2** Copia todo lo que esté contenido en el directorio dir1 al directorio dir2. Si dir2 no existe,`cp` lo creará.
- **cp -i archivo1 destino** Si la opción `-i` es especificada, el `cp` preguntará si sobreescribe "destino" en caso de que este ya exista.
- **cp -f archivo1 destino** La opción `-f` especifica que se debe sobreescribir todo sin preguntar.

Ejemplo:
% **cp .cshrc ejemplo1**

4.1.3. Moviendo archivos

mv [-if] file1 [file2 ...] destino

Este comando moverá el contenido del archivo file1 a "destino". si "destino" es un directorio, `mv` lo copiará dentro con el mismo nombre que tenía en su ubicación original. Si "destino" está en el mismo directorio que file1 `mv` funciona cambiando el nombre. Las opciones `-i` y `-f` funcionan igual que en `cp`

4.1.4. Borrando Archivos

rm [-f] [-i] archivo ...

Este comando borrará el o los archivos especificados. Las opciones `-i` y `-f` funcionan igual que en `cp`. Por omisión este comando no pide confirmación y la información eliminada por esta vía no es recuperable, por lo que se recomienda que al trabajar con información delicada se utilice la opción `-i`.

Ejemplo:
% **rm ejemplo1**

Otra forma del comando `rm` es:
rm -r dir1 ...
En este caso el `rm` borra todo el contenido del directorio dir1, incluyendo subdirectorios y archivos ocultos (que empiezan por .).

4.1.5. Visualizando el contenido de los archivos

GNU Linux presenta una serie de comandos que permiten ver el contenido de los archivos de distintas maneras. La forma más básica de desplegar un archivo es con el comando `cat`, el cual tiene la siguiente sintaxis:

- **cat archivo1** Muestra el contenido del archivo archivo1.
- **cat arch1 arch2 > arch3** Concatena o pega los contenidos de los archivos arch1 y arch2 en el archivo arch3.

Ejemplo:
% **cat prot_alinea**

. .

```
>human
VLSPADKTNV KAAWGKVGAH AGEYGAEALE RMFLSFPTTK TYFPHFDLSH GSAQVKGHGK
KVADALTNAV AHVDDMPNAL SALSDLHAHK LRVDPVNFKL LSHCLLVTLA AHLPAEFTPA
VHASLDKFLA SVSTVLTSKY RLTPEEKSAV TALWGKVNVD EVGGEALGRL LVVYPWTQRF
FESFGDLSTP DAVMGNPKVK AHGKKVLGAF SDGLAHLDNL KGTFATLSEL HCDKLHVDPE
NFRLLGNVLV CVLAHHFGKE FTPPVQAAYQ KVVAGVANAL AHKYH

>goat-cow
VLSAADKSNV KAAWGKVGGN AGAYGAEALE RMFLSFPTTK TYFPHFDLSH GSAQVKGHGE
KVAAALTKAV GHLDDLPGTL SDLSDLHAHK LRVDPVNFKL LSHSLLVTLA CHLPNDFTPA
VHASLDKFLA NVSTVLTSKY RLTAEEKAAV TAFWGKVKVD EVGGEALGRL LVVYPWTQRF
FESFGDLSTA DAVMNNPKVK AHGKKVLDSF SNGMKHLDDL KGTFAALSEL HCDKLHVDPE
NFKLLGNVLV VVLARNFGKE FTPVLQADFQ KVVAGVANAL AHRYH
%
```

Para desplegar un archivo por páginas (pantallas) se utiliza el comando `more`:
more archivo1 Muestra el contenido del archivo archivo1, una pantalla por vez.

Al ejecutar este comando, la máquina mostrará la primera pantalla del contenido del archivo y se detendrá esperando la interacción de usuario. Para mostrar el resto del contenido se puede utilizar la tecla `enter` que permite avanzar una línea a la vez o la barra espaciadora, que permite avanzar por páginas.

También se puede desplegar sólo el final o el principio de un archivo con los comandos `tail` y `head`. Por omisión, estos comandos muestran las 10 últimas líneas y las 10 primeras líneas del archivo, respectivamente

Ejemplo:

% **tail ContenidoLinuxBasico**

```
        Estados de los procesos
        Como cancelar un proceso


7. Programacion en las conchas
        Lectura y Escritura
        Estructuras de Decision
        Estructuras de Repeticion


%
```

% **head ContenidoLinuxBasico**

```
TALLER GNU Linux
Contenido

1. Sistema Operativo GNU Linux.
        Historia
        Descripcion
        Caracteristicas
        Componentes



%
```

Un comando bastante útil es el `wc`, ya que cuenta el número de líneas, el número de palabras y el número de caracteres que contiene un archivo.

Ejemplo:

wc prot_alinea

```
        13              60             648 prot_alinea
```

Esta salida quiere decir que el archivo prot_alinea tiene 13 líneas, 60 palabras y 648 caracteres.

También es importante el comando `file` el cual despliega de forma explícita el tipo del archivo que se le pasa como argumento.

Ejemplo:

% **file ContenidoLinuxBasico**

```
ContenidoLinuxBasico: International language text
%
```

4.2. Manejo de directorios

4.2.1. Visualizando el directorio de trabajo

pwd Muestra el directorio de trabajo.
Ejemplo:
% **pwd**

```
/home/ciencias/hector/public_html
```

4.2.2. Cambiando el directorio de trabajo

Para cambiarse a un directoiro se utiliza el comando `cd`

- **cd nombredir** Permite cambiarse al directorio nombredir.
- **cd** Permite cambiarse al Directorio Hogar.
- **cd ..** Permite cambiarse al directorio superior.

Ejemplo:

% **cd Fortran**

4.2.3. Creando nuevos directorios

Para crear un nuevo directorio se utiliza el comando `mkdir`

- **mkdir nombredir** Crea un nuevo directorio con el nombre nombredir.
- **mkdir -p camino1/camino2/nombredir** Crea el directorio nombredir en el camino especificado, si uno o varios de los directorios especificados en la ruta no existe, serán creados.

4.2.4. Eliminando directorios

Para borrar un directorio existen dos opciones:

- **rmdir nombredir** Elimina el directorio con el nombre nombredir, sólo si está vacío
- **rm -r nombredir** Borra el directorio nombredir, sin importar si esta vacío o no y además sin preguntar si el usuario está seguro de hacer esto o no.

Ejemplo:
% **rmdir secuencias**

4.3. Búsqueda de archivos

Para buscar archivos dentro de un árbol de directorios se utiliza el comando `find`. Dentro de las sintaxis más utilizadas están:

- **find dir -name arch -print** Busca recursivamente a partir del directorio dir el archivo arch, si lo encuentra, muestra el camino donde esta ubicado este archivo.
- **find dir -name arch -exec cmd \;**

Ejemplo:

% **find / -name ContenidoLinuxBasico -print**

```
/gil/latex/Linux/ContenidoLinuxBasico
%
```

% **find -name core -exec rm \;**

```
%
```

4.4. Manejo de permisos

GNU Linux proporciona cuentas para múltiples usuarios, asignando a cada cuenta un directorio hogar. Como se indicó en secciones anteriores, cada cuenta le es asignado un identificador numérico y un nombre (login) con los cuales obtiene acceso a la información ubicada en el directorio hogar.

El esquema de seguridad de los archivos está estructurado en tres clases de usuarios. El **dueño**(u) del archivo, el **grupo** (g) al que pertenece el dueño, y los **otros**(o) usuarios que no son el dueño o no pertenecen a su grupo. (La letra "a" se utiliza para representar a todos los usuarios: dueño, grupo y otros).

Cada archivo en GNU Linux posee un atributo para identificar el dueño y el grupo. Además, posee una serie de bits (9 en total) para definir la permisología de lectura escritura y ejecución del archivo. Estos bits están organizados como se muestra en la figura 2.

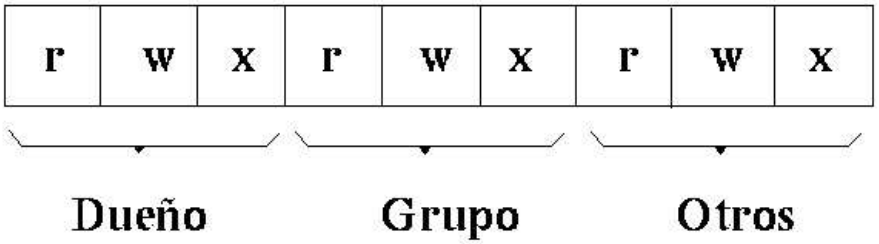

Figura 2: Organización de los bits de permisos

Con ayuda de esta estructura se puede definir, para cada archivo, una combinación de permisos para que los usuarios del sistema tengan el acceso adecuado al archivo.

El comando **ls -l** visualiza el estado actual de los permisos de un archivo. A continuación se muestra un ejemplo:
% **ls -l**

```
-rw-r--r--   1 gilberto cecalc      12601 Nov  5 14:41 PROD3.txt
-rw-r--r--   1 gilberto cecalc      17066 Nov  4 16:05 PROD6.txt
-rw-r--r--   1 gilberto cecalc      14829 Nov  6 11:09 PROD7.txt
-rwx------   1 gilberto cecalc        133 Oct 11 08:19 indice.bbl
```

```
-rwx------   1 gilberto cecalc       1017 Oct 11 08:19 indice.blg
-rwx------   1 gilberto cecalc      10757 Nov  8 11:48 indice.log
-rwx------   1 gilberto cecalc     109057 Oct  8 09:21 indice.ps
-rwx------   1 gilberto cecalc      75850 Nov  8 15:06 indice.tex
-rwx------   1 gilberto cecalc       4866 Nov  8 11:48 indice.toc
-rw-r--r--   1 gilberto cecalc       2628 Nov  8 11:44 permisos.gif
-rw-r--r--   1 gilberto cecalc     776253 Nov  8 11:45 permisos.ps
-rwx------   1 gilberto cecalc      28786 Oct 11 16:02 prueba.ps
-rwx------   1 gilberto cecalc     163455 Oct  5 09:24 tallerunix
```

La primera columna de información esta conformada por diez caracteres. El primero es una idenficación del tipo de archivo y el resto corresponde a los permisos organizados de la manera en que se muestra en la figura 2.

Para modificar los permisos de un archivo se utiliza el comando **chmod** y su sintaxis es como sigue:

chmod permisos archivos

Existen dos nomenclaturas para construir el argumentos "permisos" del comando chmod. La primera de ellas consiste en generar un decimal de tres dígitos a partir de la transformación de los tres octetos que conforman los bits de permisos. Cada grupo de tres bits representa un número binario en el rango comprendido entre cero y siete. Como base de esta primera forma de construir el argumento de permisos se asume que un uno (1) implica asignar el permido y un cero (0) significa negarlo. Si se toma un octeto cualquiera, el del dueño por ejemplo, y se le asigna permiso de lectura, escritura y se le niega el de ejecución, se tiene el número binario 110, lo cual representa al seis en decimal. El mismo procedimiento se aplica a los otros dos octetos. Asi, se puede obtener el número decimal de tres digitos que se necesita en este caso.

Ejemplo:

Asignar todos los permisos para el dueño y el grupo, y solo lectura para el resto de los usuarios de un archivo particular. En el octeto del dueño se tiene 111 lo que es igual a 7. Para el octeto del grupo se tiene el mismo valor 111, es decir, 7. Por último, en el octeto de los otros tenemos 100, es decir, 4. entonces el comando queda de la siguiente forma:

% **chmod 774 archivo**

La otra manera de construir el argumento de permisos es colocar un conjunto de caracteres que representan los permisos a ser asignados. La forma que tendría el argumento es:

ClaseDeUsuario Acción Permiso

Donde **ClaseDeUsuario** es uno o una combinación de los caracteres de la tabla 7, **Acción** es un caracter de la tabla 8 y **Permiso** es uno o una combinación de los caracteres de la tabla 9.

Cuadro 7: Clase de usuario

Caracter	**descripción**
u	dueño del archivo
g	grupo del dueño
o	los otros usuarios
a	todos los anteriores

Cuadro 8: Acciónes sobre los permisos

Caracter	**descripción**
+	asignar
-	negar
=	sobreescribir (los permisos no especificados se niegan)

El ejemplo anterior, utilizando esta nomenclatura, queda de la siguiente forma:

% **chmod ug+rwx,o=r archivo**

Cuadro 9: Tipo de permiso

Caracter	**descripción**
r	lectura
w	escritura
x	ejecución

4.5. Manejo de Medios de Almacenamiento Secundario

Los medios de almacenamiento secundario son mayormente utilizados para la elaboración de respaldos de la información contenida en los sistemas de archivos más importantes. Uno de los primeros medios utilizados fueron las cintas (tapes). GNU Linux cuenta con un comando para manipular ese tipo de dispositivos, el comando `tar` (tape archive).

El comando `tar` puede leer el contenido de una cinta:

% **tar tvf /dev/rmt1** [**archivos**]

Copiar hacia una cinta:

% **tar cvf /dev/rmt1 archivos**

Y extraer información de una cinta:

% **tar xvf /dev/rmt1** [**archivos**]

Este comando es recursivo y puede trabajar sin problemas sobre árboles completos. También puede ser utilizado sobre cualquier otro medio como por ejemplo discos flexibles.

Otra forma de utilización de este comando es empaquetar (no comprime, aunque las últimas versiones tienen esta capacidad) archivos o directorios completos en un solo archivo al cual se le coloca generalmente la extensión .tar

5. Manejo de la Red (comandos de comunicación)

5.1. TELNET

Como hemos dicho repetidas veces una de las características principales del GNU Linux es su capacidad de trabajo multiusuario. Esto hace nece-

sario un método que permita atender a varios usuarios simultaneamente. La primera solución fue hacer que cada computadora tuviese varios monitores y varios teclados. Al par monitorteclado se le llamó terminal tonto.

Con el nacimiento de las redes se creo un programa que permite emular a un terminal tonto. La ventaja de este emulador es que establece un protocolo de conexión muy sencillo por lo que se puede utilizar para conectarse a una maquina GNU Linux desde otro sistema operativo, sin embargo tiene la desventaja que, por estar emulando a un terminal tonto, no se pueden intercambiar archivos entre las máquinas que se conectan con este protocolo.

Para utilizar el Telnet se utiliza la siguiente sintaxis:

telnet [nombre de la maquina[puerto]]

Esto establece una conexión y comienza el procedimiento de login.

Otra forma de entrar al telnet es con el comando

telnet

Esto inicia el programa telnet, lo cual se reconoce por el mensaje de espera `telnet>`, en este modo el telnet acepta una serie de comandos que se pueden listar introduciendo la palabra **help** o en la págia de manual `telnet(1)`.

5.2. FTP

FTP es uno de los servicios originales de la Internet y a partir de este evolucionaron otros como el correo electrónico, las listas de discución **??** y hasta el mismo Web. FTP es el nombre que identifica dos cosas, primero al Protocolo de Transferenia de Archivos por sus siglas en inglés (**F**ile **T**ransfer **P**rotocol) y segundo al programa que establece la interfaz de usuario para este protocolo. Con el programa `ftp` es posible "copiar" archivos de un "lugar" a otro en una red. Los "lugares" pueden ser distintos directorios de una misma máquina, o máquinas diferentes ubicadas en cualquier parte dentro de la Internet.

Como el nombre lo sugiere, el `ftp` transfiere archivos de una máquina remota o "servidor" a la máquina local, en donde se encuentra el usuario (con el programa cliente).

Los sistemas operativos generalmente utilizan distintos caracteres especiales para denotar los fines de líneas, finales de archivo, etc. en los archivos de tipo texto (ascii). Para solucionar cualquier incopatibilidad entre los sistemas operativos sobre estos caracteres, se tiene un modo especial de transferencia en las aplicaciones ftp. Este modo de transferencia se le denomina modo texto o modo ascii, el cual debe ser habilitado cuando se transfieren archivos textos.

Algunas aplicaciones actuales tienen la capacidad de realizar esta operación de forma automática.

Para iniciar una sesión de `ftp` se utiliza el comando:

% ftp maquina.dominio

a lo que la máquina responderá solicitando el nombre del usuario o "login" y la palabra clave o "password" en un diálogo similar al siguiete:

```
conected to maquina.dominio

   Name (maquina:usuario): el login en la maquina remota
password required for user ******
password:
```

Una vez hecho esto aparece el mensaje de espera o "prompt" del `ftp`:

```
ftp>
```

Apartir de este momento se pueden empezar a utilizar los comandos del `ftp` para la ubicación y transferencia del o los archivos. A continuación se da una lista de los coamndos más utilizados del `ftp`:

- **cd** `[camino]`: Cambia el directorio de trabajo a el indicado por `camino`.
- **lcd** `[camino]` o **!cd** `[camino]`: Cambia el directorio de trabajo en la máquina local a el especificado por `camino`.
- **ls** `[directorio_remoto] [archivo_local]`: muestra un listados del directorio indicado por `directorio_remoto`. Si se omite la especificación de `directorio_remoto` se muestra el listado del directorio actual de trabajo en la máquina remota. Si se especifica el argumento `archivo_local` el listado se grabará en un archivo en la máquina local con el nombre indicado, en este caso el directorio no aparecerá en pantalla.
- **ls -l** `[directorio_remoto] [archivo_local]`: Al igual que en GNU Linux esta opción muestra un listado de directorio con los detalles de permisología, tamaño, dueño y fecha de creación.
- **ascii** o **as**: asigna el modo de transferencia ascii. En este modo se pueden transferir archivos cuyo contenido se encuentren en el estándar

de texto ASCII, como por ejemplo: archivo.txt (creados con `vi` del unix o con `NotePad` de MS-WinXX.

- **binary** o **bi**: Activa el modo de transferencia binario. En este modo se transfieren todos los archivos que no son tipo texto. Entre los más comunes de estos tenemos. Archivos ejecutables (.exe o .com en ms-dos y windows); archivos de procesadores de texto como los .doc; Archivos de aplicaciones como los .ppt, los .xls; las imágenes digitales .gif, .jpg, .tif, .bmp, etc; Los archivos comprimidos o empaquetados .tar, .gz, .Z, .zip, .lhz, etc. Los archivos de sonido .aiff, .au, .wav, .mpg, .mp3, etc. Los archivos de video y animación .avi, .mov, .mpg, .mpeg, .dl, etc.

- **close**: Cierra la conexión de FTP con la máquina remota sin salir de la interfaz del `ftp`. Esto permite abrir una nueva conexión con el comando **open**.

- **open** `[maquina.dominio`: Establece una nueva conexión con la máquina especificada.

- **bye** o **by**: Cierra la conexión y sale de lainterfaz del `ftp`. Tambien se utiliza para salir si la conexión ha sido cerrada con **close**

6. Comandos Avanzados

GNU Linux tiene a disposición de los usuarios una serie de herramientas que realizan tareas muy específicas. Además, presenta una característica de modularidad que hace posible combinar esas herramientas y así permitir a los usuarios ejecutar trabajos mucho más complejos.

A continuación se describen algunos de los comandos más útiles.

6.1. Ordenamiento de Archivos

sort [-t separador] [-i] archivo ...

El comando `sort` sirve para ordenar el contenido de un archivo. También tiene la capacidad de fusionar diferentes archivos en uno solo, manteniendo cierto orden en los registros.

6.2. Búsqueda de Cadenas de Caracteres en Archivos

Para buscar una cadena de caracteres dentro de uno o varios archivos se utiliza el comando `grep`

- **grep cadena arch1** Muestra las líneas del archivo arch1 que contienen la palabra cadena.
- **grep -i cadena arch1** Muestra las líneas del archivo arch1 que contienen la palabra cadena, pero sin distiguir entre mayúsculas y minúsculas.
- **grep -n cadena arch1** Muestra las líneas del archivo arch1 que contienen la palabra cadena, pero añade el número de la línea al principio

Ejemplo:
% **grep slovaca secuencias.genebank**

```
Gb_ba1:Rirrgdx  L36224 Rickettsia slovaca (strain 13-B) 16S ribosomal RNA ..
Gb_ba1:Rsu43808 U43808 Rickettsia slovaca rOmpA (ompA) gene, partial cds.
Gb_ba1:Rsu59725 U59725 Rickettsia slovaca citrate synthase (gltA) gene, p...
Gb_ba2:Rsu83454 U83454 Rickettsia slovaca rOmpA (ompA) gene, partial cds.
%
```

6.3. Cortar y Pegar Archivos

Existen comandos para extraer información desde archivos que se encuentren estructurados de forma particular. También en GNU Linux está presente un comando para poder unir información de manera sistematizada proveniente de archivos.

El primero de los comandos es **cut** el cual es capaz de cortar trozos de archivos según un patrón específico.

- **cut -cl1-l2,l3-l4,...,ln-lm archs** Este comando extrae de los archivos archs la información de cada línea comprendida entre los caracteres l1 y l2, l3 y l4 y asi sucesivamente. l1,l2,l3...lm son las posiciones de los caracteres en cada línea.
- **cut -d"sepf1,2,..,n archs** Este comando extrae de los archivos archs las columnas 1,2,..,n las cuales se encuentran separadas por el caracter sep.

Ejemplos:

% **cut -c1-10,20-30 /etc/passwd**

```
root:x:0:0ot:/bin/bas
bin:x:1:1:
daemon:x:2:/sbin:
adm:x:3:4:adm:
lp:x:4:7:lool/lpd:
sync:x:5:0in:/bin/syn
shutdown:xdown:/sbin:
halt:x:7:0in:/sbin/ha
mail:x:8:1ar/spool/ma
news:x:9:1ar/spool/ne
uucp:x:10:var/spool/u
operator:xrator:/root
games:x:12s:/usr/game
gopher:x:1er:/usr/lib
ftp:x:14:5r:/home/ftp
nobody:x:9dy:/:
gdm:x:42:4gdm:/bin/ba
xfs:x:100:t Server:/e
soffice:x:/home1/soff
yasleyda:x::/usr/peop
%
```

% **cut -d":f1,6 /etc/passwd**

```
root:/root
bin:/bin
daemon:/sbin
adm:/var/adm
lp:/var/spool/lpd
sync:/sbin
shutdown:/sbin
halt:/sbin
mail:/var/spool/mail
news:/var/spool/news
uucp:/var/spool/uucp
operator:/root
games:/usr/games
gopher:/usr/lib/gopher-data
ftp:/home/ftp
nobody:/
gdm:/home/gdm
xfs:/etc/X11/fs
soffice:/home1/soffice
```

```
yasleyda:/usr/people/yasleyda
%
```

El comando para pegar información proveniente de archivos diferentes es **paste**. Para explicar como funciona este comando supongamos que se tienen dos archivos, cada uno de los cuales contiene una columna de datos. Supongamos que el primero de estos archivos contiene las coordenadas X de cierta ubicación espacial. Ahora supongamos que el segundo archivo contine las coordenas Y y se desea mostrar por pantalla una columna al lado de la otra, entonces debemos ejecutar el comando paste como sigue:
% **paste arch1 arch2**

Donde arch1 y arch2 son los archivos que contienen la información. Este comando contiene otras opciones interesantes, revíselas con el comando man.

6.4. Comparación de Archivos

El comando diff se usa para comparar dos archivos de texto. Su función es comparar línea a línea el contenido de los dos archivos y dar como salida aquellos registros que son distintos. La sintaxis general de este comando es como se muestra a continuación:
% **diff arch1 arch2**

También puede usarse el comando sdiff que cumple la misma función que diff pero presenta la diferencia en forma horizontal:
% **sdiff arch1 arch2**

6.5. Comparación de Directorios

Este comando permite comparar el contenido de dos directorios y genera información tabulada con el resultado de la comparación. La salida de la comparación que se realiza lista el contenido de cada uno de los directorios comparados, y luego las diferencias entre el contenido de tales subdirectorios. La sintaxis de este comando es como se muestra a continuación:
% **dircmp [-d] arch1 arch2**

La opción **d** muestra el contenido donde difieren los archivos.

7. Manejo de Procesos

Ya hemos mencionado la capadidad de GNU Linux para manipular mas de un proceso a la vez. En esta sección se describen las diferentes acciones que se pueden tomar para gestionar los procesos en ejecución dentro de una máquina GNU Linux.

Para comenzar definamos primero el concepto de proceso en el marco del sistema operativo GNU Linux. Un **proceso** es un programa que se ejecuta, y al momento de ser iniciado se genera un descriptor conformado por una estructura de datos que contiene toda la información relacionada con el proceso. Esta estructura puede ser referenciada mediante un número llamado identificador de proceso (Proccess Identifier, PID). El sistema operativo mantiene una tabla con todos los procesos activos en un momento determinado la cual utiliza para la gestión de los mismos.

7.1. Estados de los procesos

Los procesos pueden pasar por diferentes estados una vez iniciados. No siempre un proceso se encuentra dentro del procesador sino que puede permanecer en otros estados mientras ocurre algún evento específico o se ejecute alguna operación sobre uno de los dispositivos periféricos del sistema.

En líneas generales los procesos en un sistema operativo multitarea como lo es GNU Linux puede encontrarse en uno de los siguientes estados. Al ser iniciado un programa este es cargado en memoria y es llevado a un estado denominado **listo** donde existe una cola donde competirá con otros procesos por el procesador. Una vez que este es despachado hacia el procesador se dice que el proceso se encuentra en estado de ejecución. El proceso estará dentro del procesador hasta que culmine o hasta que el **quantum** expire para luego regresar al estado de listo. El quantum es un tiempo que se asigna a los procesos para permanecer dentro del procesador. Si el programa se encuentra en ejecución y realiza alguna operación de entrada o salida, entonces el núcleo del sistema lo coloca en un estado **bloqueado**, donde el proceso permanecerá hasta que la operación culmine. Si la operación de entrada/salida tarda demasiado entonces el proceso es llevado a un estado llamado **suspendido-bloqueado** y al proceso se le quita todo recurso que este utilizando. Si la operación de entrada/salida culmina entonces el proceso se pasa a un estado llamado **suspendido-listo**. En este estado el proceso esta listo para competir de nuevo por el procesador pero no tiene asignado

ningún recurso del sistema. Al serle reasignados los recursos al proceso, este pasa de nuevo al estado de listo. Los estados listo, bloqueado y en ejecución son llamados estados activos; el resto son llamados estados inactivos. La figura 3 muestra las transiciones de un estado a otro.

Los procesos llamados demonios (daemons) siempre estan listos para cumplir con alguna labor, solo que si ellos permanecieran en estados activos sin hacer nada se estarian desperdiciando los recursos del sistema. Por esta razón ellos se encuentran generalmente en el estado de suspendido-listo o durmiendo.

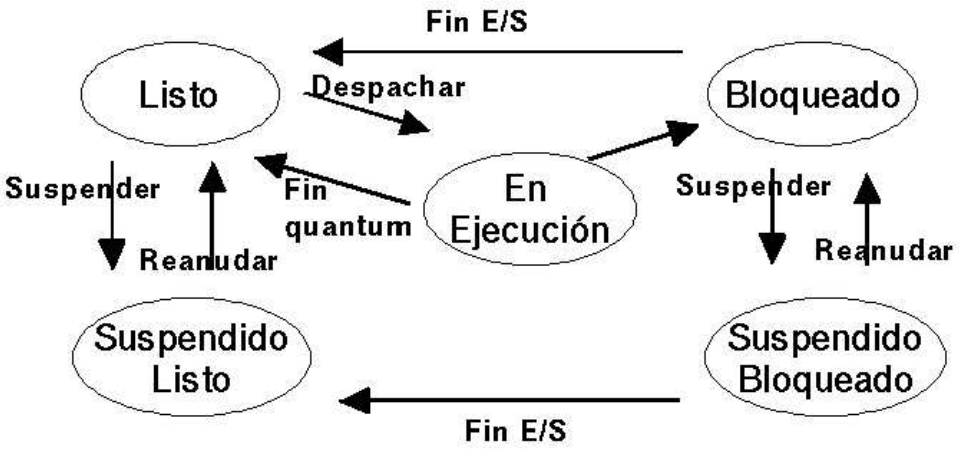

Figura 3: Estados de los procesos

Para observar el estado en que se encuentra todos los procesos del sistema se cuenta con el comando **ps**. La sintaxis de este comando en las versiones System V para desplegar una lista completa de los procesos es:

ps [-edalf]

Ejemplo:

% **ps -edalf**

```
S  UID   PID   PPID C PRI NI ADDR    SZ    STIME    TTY  TIME CMD
A  root     1   0   0  60 20 2805   344   Oct 27     -  1 2:31 init
A  root  2294   1   0  60 20 3046    84   Oct 27     -  1 9:54 syncd 60
A  root  2560   1   0  60 20 d19a   376   Oct 27     -  0:00   errdemon
A  root  3156   1   0  60 20 70ae    56   Oct 27     -  0:00   ssa_daemon

...
```

En la tabla 10 se describen algunas de las columnas que son desplegadas cuando se ejecuta el comando ps.

Cuadro 10: Información de la tabla de procesos

Columna	**símbolo**	**descripción**
PID		Número del proceso.
PPID		Número del proceso padre.
TTY		Terminal vinculado (Los demonios tendrán un ? en este campo).
S		Estado del proceso.
	O	Ejecutándose.
	R	Ejecutable en cola (Running).
	T	Detenido (sTopped).
	D	Esperando en disco.
	S	Durmiente por menos de 20 seg.
	I	Desocupado por más de 20 seg. (Sin el procesador - Inactivo).
	Z	Terminado, control pasa al padre (Zombie).
	X	Esperando más memoria.
TIME		Tiempo de procesamiento.
CMD		Comando que se ejecuta.

7.2. Cómo activar un proceso

Algunas versiones de UNIX (SunOS) introdujeron un concepto para describir un comando que se ejecuta: el concepto de tarea o trabajo (job). Un trabajo es un comando cuya ejecución se ordena desde el terminal. Un trabajo consta de uno o más procesos que se ejecutan en secuencia, bajo la tutela directa o indirecta de una sesión en la concha. Para activar un proceso entonces, la manera más sencilla es invocar su ejecución (que equivale a ejecutar un trabajo) desde la concha del sistema. La invocación consiste en escribir el nombre del archivo que contiene el código ejecutable. Al hacer ésto, la concha entenderá que debe crear un proceso hijo suyo con ese código ejecutable. Más no siempre los procesos son hijos de las conchas o creados en sesiones de usuarios. Existe un conjunto especial de procesos que no dependen de la concha, sino del proceso matríz del sistema (init). Son los llamados demonios del sistema, programas que se ejecutan constantemente y que se emplean comúnmente para atender solicitudes de servicios provenientes de los usuarios u otros programas. Los demonios son activados al encender el sistema, pero pueden reactivarse o cancelarse en cualquier momento. Volviendo con los trabajos, éstos pueden activarse .ªl frente"(foreground)., en cuyo caso la ejecución se "ve"en la pantalla del terminal; ó .ªl fondo"(background) donde el trabajo no despliega ningún mensaje directo a la pantalla. De esta forma, el usuario puede activar varias tareas, mientras que controla cuál de ellas usará la pantalla.

7.3. Manipulación de trabajos

Existe una serie de comandos que permiten gestionar los procesos en una máquina GNU Linux. Los shells cuentan con un conjunto de órdenes de control de trabajos que se puede utilizar para mover procesos de modo subordinado (background) a modo principal (foreground). La tabla 11 muestra una lista de estos comandos.

7.4. Cómo cancelar un proceso

El núcleo del sistema operativo manipula los procesos a través del envio de señales. Las señales son mecanismos de comunicación interprocesos. GNU Linux cuenta con una serie de llamadas al sistema dedicadas al manejo de señales, pero existe un comando, **kill**, que constituye una herramienta di-

Cuadro 11: Comandos para la gestión de procesos

Comando	**descripción**
CTRL-Z	Suspende el proceso actual
bg	Reanuda el trabajo parado en modo subordinado
fg	Reanuda el trabajo en modo principal
jobs	Lista todos los trabajos parados y todos los trabajos en modo subordinado.
stop	Para la ejecución del trabajo.
&	Coloca el trabajo en modo subordinado cuando este se inicia agregando este símbolo al final de la línea de comandos.

rigida al usuario no programador, que le permite el envio de señales a los diferentes procesos de los cuales él es dueño. Las señales más comunes se muestran en la tabla 12.

Cuadro 12: Señales más comunes

Señal	**Nombre**	**Descripción**
1	HUP	Reinicia el proceso.
2	INT	Interrumpe el proceso.
9	KILL	Elimina el proceso.
15	TERM	Terminación normal del proceso.

En la jerga de UNIX un usuario propietario de un proceso puede cancelar su ejecución "matando"el proceso. Quizas por ello, el comando que permite la eliminación de los procesos se le llama kill. Su sintaxis general es la siguiente:

kill [señal] PID

Ejemplo:

% **kill -9 345**

%

8. Apendice A: Programación en las conchas

Todos los shells de GNU Linux proporcionan un lenguaje de programación interpretado lo cual proporciona una herramienta muy importante que permite combinar comandos para ejecutar trabajos complejos. Esta sección pretende introducir al lector en los detalles de la programación shell de una forma resumida.

En líneas generales todo lo necesario para aprender un lenguaje es tener en cuenta los siguientes puntos: cómo se manejan las variables, cómo leer, cómo escribir, manejo de decisiones y manejo de lazos. La mayoria de los lenguajes presentan estas funcionalidades aparte de otras capacidades propias que puedan tener. En lo que sigue se describen brevemente los puntos anteriores.

8.1. Estructura de un script

Cualquier programa shell puede ser introducido directamente sobre la línea de comandos. Cada vez que se introduzca una línea de código, esta será interpretada y ejecutada inmediatamente. Por comodidad, se puede escribir el código en un archivo texto, darle permiso de ejecución y luego correrlo como cualquier otro comando. A esos archivos texto se les llama **scripts**.

Por la presencia de diversas conchas y sintaxis de programación diferentes, se hace necesario distinguir dentro de los scripts el tipo de concha que debe utilizarse para correr un script. Para tal fin, la primera linea del programa indica cual es el tipo de shell. La sintaxis de esta línea es la siguiente:

#!/bin/concha

Donde los posibles valores de "concha" pueden ser: sh, csh, ksh, bash, tcsh, zsh. Lo cual distingue cual será la concha utilizada para interpretar los comandos del script.

Después de esta linea lo que sigue son las intrucciones del programa La mayoria de las veces estas intrucciones están conformadas por los comandos de GNU Linux, lo cual brinda la oportunidad de ejecutar comandos por lotes.

La diferencia entre la programación de las distintas conchas se basa en la sintaxis de la manipulación de variables, las estructuras de decisión y las estructuras de repetición. En este curso solo mostraremos la programación en Bourne Shell, la cual sirve también para las conchas Korn y Bash.

8.2. Manipulación de variables

En la programación shell las variables no poseen tipo y no es necesario la declaración de estas. Para asignar cualquier valor a una variable basta con ejecutar una instrucción como la que sigue:

VARIABLE=valor

Por convención el nombre de las variables en shell siempre es colocado en mayúsculas. Para acceder al valor de una variable se debe hacer referencia a esta de la siguiente forma:

$VARIABLE

Las variables en shell pueden comportarse como listas de valores. La asignación se hace del modo siguiente:

VARIABLE="valor$_1$ valor$_2$ ... valor$_n$"

Es posible almacenar la salida de un comando en una variable. Esto se hace de la siguiente manera:

VARIABLE=‘comando‘

8.3. Lectura y Escritura

Para leer datos desde el teclado y colocarlos como contenido de una variable se utiliza un conjunto de caracteres especiales ($<), su sintaxis es como sigue:

read VARIABLE

La escritura por pantalla de cualquier texto se realiza con la ayuda del comando echo.

echo comentario

Las conchas GNU Linux utilizan un conjunto de caracteres especiales para realizar funciones como secuenciamiento, encauzamiento, comodines, etc. Estos caracteres son llamados **metacarateres**. Si se desea imprimir algún metacaracter entonces hay que utilizar caracteres especiales que convierten los metacaracteres en caracteres ordinarios. La tabla 13 muestra una lista de los metacaracteres y a su vez los caracteres especiales que los convierten en ordinarios.

Cuadro 13: Metacaracteres

Caracter especial	**Descripción**
’	Elimina el significado especial de ’ < > # * ? & | ; ( ) [ ] ^ , espacios en blancos, tabs.
”	Igual que ’ excepto para $ ‘ ”\
\	Elimina el significado especial del caracter que lo siga.

8.4. Manipulación de parámetros en los programas

Ya hemos visto como referenciar el valor de una variable a través del símbolo $. Ahora veremos como manipular cada uno de los parámetros usados al invocar un programa shell: $1,$2, ...$9. Cada uno de ellos permite referenciar a los parámetros 1,2 ...9 respectivamente. El símbolo $* hace referencia a la lista de parámetros completa. Con $# se puede obtener el número de parámetros que conforman una lista o línea de comandos. Esta información puede ser útil cuando se escribe un programa shell que requiera un número exacto de parámetros.

8.5. Estructuras de Decisión

En Bash se puede utilizar estructuras de decisión como sigue:

```
if [ decision ]
then
   comandos
```

```
else
   comandos
fi
```

También está disponible una estructura de decisión mútiple:

```
case $VAR
in
   valor1)
      comando1
      comando2;;
     .
     .
     .
   valorn)
      comando1
      comando2;;
   *)
      comando1
      comando2;;
esac
```

La tabla 14 muestra los caracteres de comparación lógica que pueden ser utilizados dentro de la decisión de las sentencias de decisión y los lazos de repetición que se verán más adelante.

8.6. Estructuras de Repetición

8.6.1. Lazo while

Una de las estructuras de repetición presentes en la programación Bash es el lazo **while**

```
while [ Condicion ]
do
   comandos
done
```

Ejemplo:

Cuadro 14: Caracteres de comparación lógica

Símbolo	Descripción
-eq	igual que
-ne	diferente que
-gt	mayor que
-lt	menor que
-ge	mayor igual
-le	menor igual
-a	Y lógico
-o	O lógico

```
#!/bin/sh

CONT=1

while [ $CONT -le 1000 ]
do
   echo $CONT
   CONT=‘echo $CONT + 1 | bc‘
done
```

8.6.2. Lazo for

El lazo **for** en shell tiene una forma diferente de trabajar, pero le da mayor versatilidad . Esta sentencia trabaja sobre listas, y en cada iteración la variable de control o contador conteniene un elemento de la lista. Ejemplo

```
for i in elemento1 elemento2 elemento3 .... elementoN
do
   comandos
done
```

En la primera iteración la variable de control “i” tendrá como valor el elemento1, en la segunda iteración tendrá elemento2, y así sucesivamente hasta elementoN.

La lista puede estar conformada por la salida de un comando, Por ejemplo:

```
for i in ‘cmd‘
do
   comandos
done
```

En este caso la variable “i” tendrá en cada iteración un elemento de la salida del comando “cmd”.

Ejemplo:

```
for i in ‘ls‘
do
   echo $i
done
```