Sistema Operativo Linux

Gilberto Díaz

10 de septiembre de 2018

Licencia de Uso

Índice General

Índice de Figuras

Índice de Tablas

Acerca de este manual

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 Unix/Linux. Asi mismo, los introduce en los detalles de la administración de este tipo de sistemas operativos. Intenta igualmente ser un tutorial organizado por tipo de tarea.

Objetivos

Este material trata sobre el uso, manejo y administración del sistema operativo Unix Incluye ejemplos prácticos basados en GNU/Linux

Al finalizar este manual usted debe estar en capacidad de:

- Acceder a su ambiente de trabajo en una máquina GNU Linux.
- Entender y utilizar los distintos elementos del sistema operativo.
- Entender y utilizar los conceptos de directorios y archivos.
- Ejecutar aplicaciones.
- Utilizar aplicaciones de red para comunicarse con otros sistemas
- Instalar sistemas Unix basados en Linux.
- Conocer los detalles de arranque y detención de Unix.
- Manejar usuarios.
- Administrar sistemas de archivos.
- Administrar procesos y aplicaciones.
- Administrar interfaces de red.
- Manejar sistemas de impresión.

- Conocer los diferentes detalles de seguridad.
- Conocer herramientas para supervisión de redes.

Organización

Este manual está organizado en 5 capitulos 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**
- **Manejo 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.
- **Manejo de Directorios** En esta seccion se demuestra la forma en la que el usuario debe ver el contenido y propiedades de los directorios ademas de crear nuevos directorios asi como tambien eliminarlos, en el mismo orden de ideas el usuario podra a su vez ser capaz de buscar archivos y manejar la permisologia de los mismos.
- **Manejo de medios de almacenamiento secundarios** Se muestra el uso correcto de los distintos medios de alamacenamiento secundarios como Pen Drives y Discos Compactos, ademas de compartir los Directorios en red .

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.
`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.
`alt-x`	La secuencia de caracteres `alt` delante de una letra indica que se debe mantener presionada la tecla **alt** al mismo tiempo que se presiona la letra indicada.

Capítulo 1

El Núcleo de Linux

1.1. Introducción

El Núcleo (*kernel*) de Linux funciona como parte de un sistema que es de gran utilidad visto como un todo. Es por esto que comenzaremos por contextualizarlo dentro del sistema operativo. Según la descomposición en subsistemas propuesta por Garlan 1994, el sistema operativo linux está compuesto por cuatro grandes subsistemas:

- Aplicaciones de usuario.
- Servicios del Sistema Operativo.
- Kernel Linux.
- Controladores de Hardware.

Figura 1.1: Descomposición del Sistema Operativo Linux en capas

Cada capa es un subsistema y solo puede comunicarse con las capas adyacentes a ella. Adicionalmente, la dependencia entre capas viene dada de arriba a abajo, es decir, las capas superiores dependen de las capas inferiores, mientras que los subsistemas más bajos no dependen de las capas superiores.

El núcleo de Linux es el programa que se encarga de la gestión de todos los recursos de hardware y software presentes en un computador. Éste presenta una interfaz virtual a los procesos del usuario. Los procesos son escritos sin necesidad de conocimiento alguno sobre el tipo de Hardware instalado en un computador. Además, Linux soporta el multiprocesamiento de tal manera que se hace transparente a los procesos del usuario: cada proceso actúa como si fuera el único proceso corriendo en el computador, con uso exclusivo de los recursos del sistema, por ejemplo, memoria principal. Por lo tanto el kernel también es responsable de mediar el acceso a los recursos de hardware del sistema de tal manera que cada proceso tenga un acceso justo a ellos y además, debe mantener la seguridad inter-procesos adecuada para su correcto desempeño [6].

El núcleo de Linux se encarga de:

- Gestión de la memoria: el núcleo asigna memoria a otros programas que se encuentran en ejecución.
- Gestión del procesador: maneja las colas de los procesos y sus prioridades.
- Administración de los periféricos: gestiona el uso de todos los dispositivos periféricos del sistema tales como: dispositivos USB, ratón, teclado, monitor, etc.
- Manejo de medios de almacenamiento: organiza la información en dispositivos como discos duros, unidades de CD, floppies, cintas, pen drives, etc.
- Comunicación con otros sistemas: gestión de interfaces de red, etc.

1.2. Arquitectura

De acuerdo a [3], el núcleo de Linux está compuesto por cinco subsistemas principales:

- El Planificador: Es el responsable de controlar el acceso de los procesos al procesador. El planificador rige una política que asegura que los procesos tendrán un acceso justo al procesador, mientras que a su vez debe asegurarse que el kernel está desempeñando las acciones pertinentes al hardware necesarias para la ejecución del proceso.
- El Manejador de Memoria: Permite que la memoria principal del sistema sea compartida de manera segura por multiples procesos. Adicionalmente, el manejador de memoria soporta la llamada Memoria Virtual, ésta permite que Linux soporte procesos que utilizan más memoria de la que dispone el sistema.

- El Sistema Virtual de Archivos: Abstrae los detalles de los diferentes dispositivos de hardware presentando una interfaz de archivos común a todos los dispositivos. Además, el sistema virtual de archivos soporta diversos formatos de archivos del sistema que son compatibles con otros sistemas operativos.
- La Interfaz de Red: Provee acceso a diversos estandares de red y a una gran variedad de hardware de red.
- La comunicación inter-procesos: Este subsistema soporta diversos mecanismos para la comuicación proceso a proceso en un mismo sistema Linux.

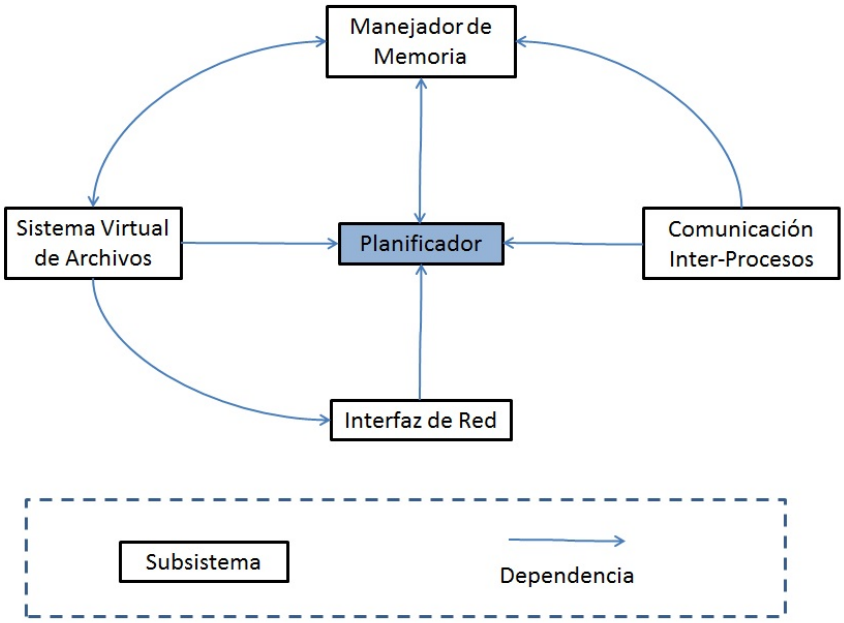

Figura 1.2: Descomposición en subsistemas del Kernel Linux

Capítulo 2

Tareas Básicas

En Linux, al igual que en otros sistemas Unix, se cuenta con distintos ambientes de trabajo. En este capítulo estudiaremos como realizar tareas comunes tales como: gestión de archivos, gestión de directorios, etc. utilizando interpretadores de comandos y ambientes de ventanas.

2.1. Línea de comandos (Shells)

Las conchas o *shells* son los programas de 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 sólo comando.

2.1.1. 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 omisión, 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.

2.1.2. Archivos de configuración de los shell

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.

2.1.3. Características de los Shells

Los shells 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 los shells 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:** Un shell 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 el shell).

- **Interpretador de lenguaje de programación:** Los shells "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 del shell Bash.

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 los shells 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 los shells *Bash* se muestran en la tabla 2.1.

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 del shell
`env`	Establece y muestra una variable de ambiente
`source`	Ejecuta los comandos escritos en un archivo. Puede ser utilizado para actualizar el ambiente del shell
`time`	Muestra el tiempo de ejecución de un comando
`unalias`	Elimina una definición de un alias
`unset`	Elimina una variable del shell
`unsetenv`	Elimina una variable de ambiente

Tabla 2.1: Comandos de los shells tipo Bash

El segundo tipo de comando está conformado por una serie de programas cuyo comportamiento es independiente del tipo de shell, 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, 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.
     ....
```

2.1.4. Gestión de archivos

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
```

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**

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`

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 .).

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 Shell
        Lectura y Escritura
        Estructuras de Decision
        Estructuras de Repeticion


%
```

% **head ContenidoLinuxBasico**

```
TALLER GNU Linux
Contenido

1. Sistema Operativo 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
%
```

Manejo de directorios

Visualizando el directorio de trabajo

pwd Muestra el directorio de trabajo.

Ejemplo:

% **pwd**

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

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**

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.

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**

2.1.5. 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 \;**

```
%
```

2.1.6. 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 **??**.

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 **??**.

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 sólo 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 2.2, **Acción** es un caracter de la tabla 2.3 y **Permiso** es uno o una combinación de los caracteres de la tabla 2.4.

Tabla 2.2: Clase de usuario

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

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

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

Tabla 2.3: Acciónes sobre los permisos

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

Tabla 2.4: Tipo de permiso

Caracter	descripción
r	lectura
w	escritura
x	ejecución

2.1.7. 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

2.2. Ejemplos

El comando **touch** actualiza la hora y fecha de un archivo. Si el archivo no existe entonces lo crea vacío. Ejecute el siguiente comando:

```
touch datos1
```

Revise los permisos del archivo

```
ls -l datos1
```

Modifique los permisos para que el dueño tenga todos los permisos, el grupo sólo lectura y ejecución y el resto de usuarios sólo lectura

```
chmod 754 datos1
```

Agregue contenido a un archivo desde la línea de comandos

```
echo "Taller de Linux" > linux
```

Verifique el contenido del archivo

```
cat linux
```

Capítulo 3

Tareas Avanzadas

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.

3.1. Comandos Avanzados

En esta sección se describen algunos de los comandos más útiles en el uso de la línea de comandos.

3.1.1. 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.

3.1.2. 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.
%
```

3.1.3. Cortar y Pegar Archivos

Existen comandos para extraer información desde archivos que se encuentren estructurados de forma particular. También en 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.

3.1.4. 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**

3.1.5. 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.

3.2. Manejo de Procesos

Ya hemos mencionado la capadidad de 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 Linux.

Para comenzar definamos primero el concepto de proceso en el marco del sistema operativo 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.

3.2.1. 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 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 3.2 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.

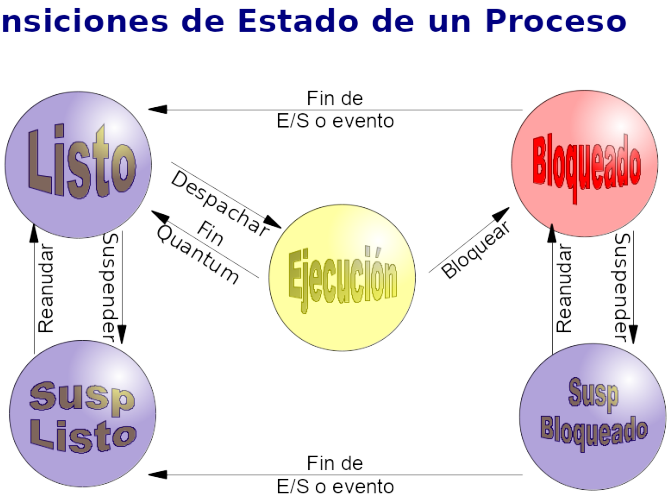

Figura 3.1: Estados de un Proceso

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 3.1 se describen algunas de las columnas que son desplegadas cuando se ejecuta el comando ps.

Tabla 3.1: 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.

3.2.2. 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.

3.2.3. Manipulación de trabajos

Existe una serie de comandos que permiten gestionar los procesos en una máquina 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 3.2 muestra una lista de estos comandos.

Tabla 3.2: 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.

3.2.4. 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. Linux cuenta con una serie de llamadas al sistema dedicadas al manejo de señales, pero existe un comando, **kill**, que constituye una herramienta dirigida 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 3.3.

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:

Tabla 3.3: 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.

kill [señal] PID
Ejemplo:

% **kill -9 345**

%

3.3. Ejemplos

Vea cual es el nombre del usuario que está utilizando

```
echo \$USER
```

Muestre los procesos que está ejecutando usted

```
ps aux | grep \$USER
```

Extraer los identificadores de los procesos que usted está ejecutando

```
ps aux | grep \$USER | tr -s ' ' | cut -d" " -f2
```

Capítulo 4

Programación Shell

Todos los shells de 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.

4.0.1. 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 diversos shells y sintaxis de programación diferentes, se hace necesario distinguir dentro de los scripts el tipo de shell 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/shell

Donde los posibles valores de "shell" pueden ser: sh, csh, ksh, bash, tcsh, zsh. Lo cual distingue cual será el shell utilizado 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 Linux, lo cual brinda la oportunidad de ejecutar comandos por lotes.

La diferencia entre la programación de las distintos shells 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 los shells Korn y Bash.

4.0.2. 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‘

4.0.3. 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

Los shell en 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 4.1 muestra una lista de los metacaracteres y a su vez los caracteres especiales que los convierten en ordinarios.

Tabla 4.1: 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.

4.0.4. 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.

4.0.5. 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 4.2 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.

4.0.6. Estructuras de Repetición

Lazo while

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

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

Tabla 4.2: 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

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
```

4.1. Ejemplos

Edite un archivo de nombre `contar.sh` y agregue las siguientes líneas

```
\begin{verbatim}
     #!/bin/bash

     CONT=1

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

\end{verbatim}
```

Asigne permisos de ejecución

```
chmod +x contar.sh
```

Ejecute el siguiente comando para ejecutar el script anterior

```
./contar.sh
```

4.2. Ejercicios

1. Escriba un script que cree un archivo con los identificadores de los procesos que usted está ejecutando
2. Escriba un script que cuente el número de líneas de un archivo
3. Escriba un comando que encuentre todas las imágenes .jpg en su directorio hogar.

Bibliografía

[1] Kernel.org.

[2] Linux.die.net.

[3] Ivan Bowman. Conceptual architecture of the linux kernel.

[4] Marco Cesati Daniel P. Bovet. *Understanding the Linux Kernel*. O'Reilly & Associates Inc., 2003.

[5] Michael K. Johnson. Diff, Patch, and Friends. *Linux J.*, 1996(28es), 1996.

[6] M. Tim Jones. Anatomy of the linux kernel.

[7] M. Tim Jones. Inside the linux boot process.

[8] Robert Love. *Linux Kernel Development (3rd Edition)*. Addison-Wesley Professional, 3 edition, 7 2010.

[9] Gary Lawrence Murphy. *The Linux Kernel: Blueprints for World Domination*. MacMillan Computer Publishing, 2000.

[10] Peter Jay Salzman, Michael Burian, and Ori Pomerantz. *The Linux Kernel Module Programming Guide*. CreateSpace, 1 2009.