



Laboratorio Docente de
Computación



Curso de Usuario Linux

<ldc@ldc.usb.ve>

Mayo 2006

Índice

1. Historia de UNIX	4
2. Qué es Linux	5
2.1. Características	5
3. Distribuciones Linux	7
4. Conexión, Desconexión y Ambiente Gráfico	8
4.1. Conexión y Desconexión	8
4.2. Ambiente Gráfico	9
5. Intérprete de Comandos	9
6. Cómo acceder a la documentación	10
7. Estructura del Sistema de Archivos	10
8. Rutas de acceso y Enlaces	11
8.1. Rutas de Acceso	11
8.2. Enlaces	12
9. Comandos Básicos	12
9.1. Manipulación de Directorios	12
9.2. Manipulación de Archivos	13
9.3. Visualización de Contenido de Archivos	15
9.4. Edición de Archivos	16
9.5. Impresión	17
9.6. Control de Procesos	18
10. Archivos, acceso de usuarios y protección	19
10.1. Tipos de Archivo	19
10.2. Control de Acceso y Modos de Protección	20
11. Expresiones Regulares	22
11.1. Uso de <i>grep</i>	23
12. Redireccionamiento de Entrada, Salida y Error Estándar	28
13. Pipes	29
14. Variables de Ambiente	29

15.Comandos para compresión de Archivos	31
16.Respaldo y Recuperación de Archivos: <i>tar</i>	34
17.Secure Shell: SSH	35
17.1. Características	35
18.Programacion con el Shell “Shellscripting”	36
18.1. Declaración de variables	38
18.2. Test y Comparaciones	40
18.3. Ciclos	43
19.Ejercicios	45

1. Historia de UNIX

En los años setenta, dos investigadores de los Laboratorios Telefónicos Bell (Bell Telephone Labs o BTL) llamados Dennis Ritchie y Ken Thompson desarrollaron un sistema operativo muy elegante al que llamaron Unix. Eligieron el nombre Unix como una burla al proyecto en el que habían trabajado anteriormente: Multics. Al completar el desarrollo de Unix, Ritchie y Thompson expusieron su diseño en una conferencia internacional donde varios de los participantes les pidieron una copia de este sistema. La presión de los investigadores en obtener una copia de Unix motivó a los ejecutivos de BTL a licenciar su uso como una herramienta de investigación. La licencia de Unix era muy barata para las universidades y bastante cara para la industria.

Una de las universidades que adquirió una licencia de Unix fue la Universidad de California en Berkeley. La motivación principal era poder experimentar con el primer sistema operativo que incluía código fuente. Al poco tiempo, la gente de Berkeley había escrito varios programas utilitarios para Unix que otros investigadores podrían encontrar útiles. La Universidad decidió entonces distribuir este código a la comunidad y le llamó a sus distribuciones BSD (Berkeley Software Distribution). A pesar de que al principio las distribuciones de Berkeley consistían principalmente en herramientas para los usuarios, muy pronto comenzaron a cambiar la forma en que el propio sistema operativo funcionaba. Implementaron el manejo de memoria virtual y programaron el soporte para los protocolos del Arpanet que luego se convertiría en el conocido Internet.

A mediados de los años ochenta, Richard Stallman, entonces en el Instituto Tecnológico de Massachussets (MIT), decidió dedicarse a la construcción de lo que denominó software libre. El razonamiento de Stallman era que los mayores progresos en la industria del software surgen cuando se coopera entre programadores. Según Stallman, las industrias de la época estaban atentando contra la libertad de los usuarios y programadores de compartir el software, así que decidió crear un movimiento de programadores orientados a este fin. A este sistema le llamó GNU, un acrónimo recursivo que significa Gnu's Not Unix (GNU no es Unix).

Para las personas deseosas de correr Unix en las ahora populares PCs, quedaba únicamente una alternativa, Minix. Minix era un sistema operativo parecido a Unix desarrollado por el Profesor Andrew Tanenbaum para enseñarle a sus alumnos acerca del diseño de sistemas operativos. Sin embargo, debido al enfoque puramente educacional de Minix, Tanenbaum no permitía que este fuera modificado demasiado ya que esto complicaba el sistema y no permitía que sus estudiantes lo entendieran en un semestre.

Un estudiante de Finlandia, Linus Torvalds, al ver que no era posible

extender Minix, decidió escribir su propio sistema operativo compatible con Unix. Miles de personas que querían correr Unix en sus PCs vieron aquí su única alternativa debido a que a Minix le faltaban demasiadas cosas y BSD, a pesar de tener toda la funcionalidad esperada, tenía problemas legales. El proyecto GNU que Stallman había iniciado hacía ya casi diez años había producido para este entonces un sistema casi completo a excepción del kernel, que es el programa que controla el hardware de la máquina. Torvalds decidió utilizar el casi completo sistema GNU y completarlo él mismo con su propio kernel, al resultado le llamó Linux. Richard Stallman insiste aún en que el sistema debiera ser llamado GNU/Linux, ya que incluye más código del proyecto GNU que del proyecto Linux.

2. Qué es Linux

Linux es un sistema operativo, creado por miles de programadores en todo el mundo que, provee al usuario de una interfaz para interactuar de algún modo con el software y hardware instalado en la maquina. Linux es únicamente su kernel (núcleo). Existe un mito muy extendido de que Linux es también el conjunto de aplicaciones que trabaja sobre este núcleo y que han sido compiladas para el mismo, sin embargo esto no es así.

Fue creado en el año 1991 por un estudiante de la Universidad de Helsinki llamado Linus Benedict Torvalds. Surgió a raíz de un experimento para descubrir las posibilidades del procesador 80386 y fue basado en el sistema operativo experimental Minix que fue confeccionado por Andrew S. Tannenbaum. Posteriormente Linus Torvalds reescribió desde cero todo el sistema dejando de lado a Minix y así aprovechó toda la nueva arquitectura de 32 bits para su S.O.

La parte central de Linux (conocida como núcleo o kernel) se distribuye a través de la Licencia Pública General GNU, lo que básicamente significa que puede ser copiado libremente, cambiado y distribuído, pero no es posible imponer restricciones adicionales a los productos obtenidos y, adicionalmente, se debe dejar el código fuente disponible, de la misma forma que está disponible el código de Linux. Aún cuando Linux tenga registro de Copyright, y no sea estrictamente de dominio público. La licencia tiene por objeto asegurar que Linux siga siendo gratuito y a la vez estandar.

2.1. Características

- Multitarea, varios programas ejecutándose “simultáneamente”.
- Multiusuario, varios usuarios en la misma máquinas al mismo tiempo.

- Multiplataforma, funciona con la mayoría de las plataformas del mercado: Intel 386/486/Pentium, Motorola 680, Sun Sparc.
- Tiene protección de la memoria entre procesos, de manera que alguno de ellos no pueda colgar el sistema.
- Soporte para varios sistemas de archivo comunes, los más conocidos son ext2, ext3, reiserfs, xfs y muchos otros.
- Incluye TCP/IP, incluyendo ftp, telnet, NFS.

3. Distribuciones Linux

Durante los últimos años varios grupos motivados con la idea del Software Libre han desarrollado diversas distribuciones (coloquialmente llamadas Distros) basadas en Linux. En principio usan los mismos principios y respetan los estándares establecidos pero presentan nuevas funcionalidades o hacen hincapié en mejorar ciertos aspectos. Las más conocidas son:

- **Debian GNU/Linux.**

Debian es un Sistema Operativo libre que provee más de 8.710 packages, precompilados y armados en un buen formato para fácil instalación. Fue pionero en las instalaciones de paquetes por red usando repositorios en la Internet. Debian es mantenido por casi 1.000 desarrolladores activos en todo el mundo quienes aportan su conocimiento de manera voluntaria.

- **Red Hat Enterprise Linux.**

Red Hat Enterprise Linux es la plataforma líder para la comunidad de Software Libre. Es vendido por suscripción, y está certificado por los mejores desarrolladores de hardware y Software. Enterprise Linux integra la innovación de la tecnología de la comunidad del Software Libre y la estabilidad de una verdadera plataforma de calidad.

- **Fedora Project.**

Fedora es un proyecto de código abierto patrocinado por Red Hat y mantenido por la comunidad Fedora. Provee nuevas tecnologías que eventualmente serán parte de los productos de Red Hat. Brinda al usuario herramientas que facilitan el mantenimiento y configuraciones del sistema.

- **Gentoo Linux.**

Es una distribución especial de Linux ya que puede ser automáticamente optimizada y personalizada para cualquier aplicación o necesidad. Altamente configurable, de buen desempeño y adaptabilidad, gracias a una tecnología llamada Portage, Gentoo Linux puede convertirse en el servidor ideal, la estación de trabajo de un desarrollador, estación de uso profesional, sistema de juegos, etc.

- **Mandriva Linux.** Anteriormente conocido como Mandrake Linux, es un OS amigable al usuario que se especializa en la facilidad de uso. Está libremente disponible en muchos idiomas en el mundo. El PowerPack

de Mandriva Linux contiene más de 2.300 aplicaciones de alta calidad incluyendo la suite completa de Office, además ofrece soporte para su instalación, todo por un costo 10 veces menor al paquete Microsoft Windows + MS Office.

- **Slackware.**

Es la distribución oficial de Patrick Volkerding, es un sistema avanzado diseñado con las prioridades de fácil uso y estabilidad. Incluye lo último en software mientras que mantiene ciertos aspectos tradicionales de UNIX. Contiene un programa de instalación de fácil uso, mucha documentación en línea y gran variedad de paquetes.

- **SuSE Linux.**

SuSE Linux Professional provee características necesitadas para PC personales y Servidores. SUSE Linux Professional también incluye más de 1.000 aplicaciones de código abierto líderes en el mundo.

- **Ubuntu.**

Es un Sistema operativo basado completamente en Linux derivado de Debian libremente distribuido y gratis. Ubuntu está disponible para computadores de escritorio y servidores. Las versiones actuales son compatibles con diversas arquitecturas como: PC Intel x86, 64-bit PC (AMD64) y PowerPC (Apple iBook y Powerbook, G4 y G5).

4. Conexión, Desconexión y Ambiente Gráfico

4.1. Conexión y Desconexión

Este aspecto se refiere al inicio y finalización de una sesión de usuario. Una vez que el sistema ha arrancado correctamente, el proceso de inicialización se detiene en una pantalla de control de acceso, comúnmente llamada **Inicio de Sesión**. En el inicio de sesión de usuario se deben introducir el nombre del usuario y la contraseña que deberá existir en el sistema Linux al cual se desea ingresar, de otra manera dará un error de nombre de usuario o contraseña incorrecta. El sistema consulta en los archivos de gestión de usuarios para verificar la información introducida, si ésta es correcta, entra en la sesión del usuario y se podrá acceder a un intérprete de comandos. Igualmente el usuario podrá cerrar sesión en cualquier momento haciendo ejecutando el comando *logout* o secuencia especial CTRL+D.

4.2. Ambiente Gráfico

El entorno gráfico en Linux está representado por el sistema X Window, desarrollado originalmente en el MIT (Massachusetts Institute for Technology) alrededor de 1987, el objetivo era crear un entorno gráfico de usuario que pudiera utilizarse en múltiples plataformas de forma flexible y sin grandes dependencias de hardware.

El aspecto y comportamiento de cada uno de los clientes X se ve parcialmente influido por el administrador de ventanas. Éste es el que determina cómo modificar el tamaño de la ventana, el aspecto del marco de la ventana, etc. Linux suministra diversos administradores de ventanas, lo más usados son KDE, GNOME, XFCE, fvwm, y muchos otros. La ventaja de este sistema es que cada usuario puede tener distintos aspectos y configuraciones para su ambiente.

En el inicio de sesión que se ejecuta en un ambiente gráfico, el sistema presenta una pantalla gráfica donde el usuario colocará el nombre de usuario y contraseña proporcionado por el administrador del sistema, una vez revisados los archivos correspondientes para verificar que los datos son correctos y permitir el acceso, en caso de haber algún error, el manejador gráfico indicará un error de nombre de usuario o contraseña incorrecta. Para salir del manejador, éstos proveen una opción de menú que le permite al usuario cerrar su sesión, lo cual retornará a la pantalla de inicio.

Asimismo, la sesión que provee el manejador de ventanas ofrece un menú de inicio donde se encuentran las aplicaciones instaladas, un escritorio con enlaces a programas, archivos o carpetas, entre otras funcionalidades que dependen del manejador de ventanas seleccionado.

5. Intérprete de Comandos

El usuario tendrá disponible un Shell (o terminal) de comandos donde podrá comunicarse con el núcleo del Sistema Operativo. Un **Shell** o Intérprete de Comandos es un programa de servicio que envía comandos de la forma correcta al núcleo (o kernel). El shell protege lógicamente del usuario el núcleo y el resto de programas basados en él.

En el pasado, el shell más conocido y más distribuido con los sistemas UNIX comerciales era el shell Bourne, desarrollado por Steven Bourne. Existe desde mediados de la década de los setenta. Más tarde se desarrollaron otros para los distintos derivados de UNIX. Entre ellos, los más conocidos son el C Shell y Korn Shell. Éste último incluye todas las posibilidades del shell Bourne e incorpora otras nuevas que hacen el trabajo más cómodo.

Linux suministra a sus usuarios distintos tipos de intérpretes. El conocido como Bourne-Again-Shell (abreviado *Bash* es el más utilizado e incorpora todos los aspectos positivos de los shells Bourne y Korn. Otros conocidos son: el shell de Kenneth Almquist (*ash*), el shell Korn de dominio público (*pksh*), *tsh* como nuevo desarrollo del Shell C y el shell z (*zsh*) de Paul Falstad.

Para cerrar un Interpretador de comandos se utiliza el comando *logout*, *exit* o con la combinación de teclas **CTRL-D**.

6. Cómo acceder a la documentación

Debido al auge de Linux en el mundo, existen en la Internet muchos sitios que proveen documentación. Igualmente muchos libros publicados sobre diversos tópicos, pero la herramienta de información más poderosa que contienen los sistemas Linux son las páginas del manual, ésta provee una referencia a los comandos respecto al modo de uso y las opciones que presentan. Se accede a un terminal a través del comando: *man nombre_Comando*, por ejemplo:

```
[user@host]:> man ls
NAME
    ls - list directory contents
SYNOPSIS
    ls [OPTION]... [FILE]...
DESCRIPTION
    ...
```

7. Estructura del Sistema de Archivos

La estructura del sistema de archivos en Linux está representada en un árbol de directorios o jerárquico. Por sistema de archivos se entiende la forma en que el sistema operativo administra los datos.

La unidad más pequeña del sistema de archivos es el archivo. Se diferencian entre distintos tipos de archivos como regulares (texto plano o binarios), directorios y de dispositivos.

Se han creado muchas implementaciones para el Sistema de Archivo de Linux (y UNIX en general), estos se diferencian en detalles de implementación y nuevas funcionalidades que se agregan. Los más comunes son ext2, ext3, xfs, reiserfs, entre otros. En cuanto a los directorios estándar que determinados programas y funciones usan podemos mencionar:

- `/`. La raíz del sistema de archivos.
- `/bin`. Este directorio contiene binarios utilizados por todos los usuarios.
- `/usr`. Este directorio contiene los programas usados por los usuarios, en sus subdirectorios se pueden conseguir librerías, páginas del manual y los binarios.
- `/etc`. Contiene los archivos para la configuración necesarios para la administración del sistema. La ventaja de linux es que cualquier utilidad es configurable mediante un simple archivo de texto.
- `/sbin`. Contiene los comandos del administrador o root del sistema.
- `/tmp`. Algunos programas crean archivos temporales que desaparecen al finalizar la ejecución del comando. Son guardados aquí.
- `/dev`. Usado para los dispositivos periféricos y otros componentes de hardware vinculados al sistema operativo.
- `/root`. Es el directorio principal del usuario root.
- `/home`. Por estándar los directorios principales de todos los usuarios comunes se encuentran aquí.

8. Rutas de acceso y Enlaces

8.1. Rutas de Acceso

Una administración de archivos correcta se basa en el principio de que todos los archivos deben poder identificarse unívocamente. Bajo Linux, este principio se cumple gracias a los nombres de rutas. El nombre de ruta de un archivo representa otra forma de descripción de los pasos para encontrar un archivo. El directorio raíz se representa mediante la barra diagonal (`/`). A continuación se incluyen los siguientes directorios del nombre de ruta separados entre sí mediante una barra diagonal. Ésta tiene una doble función: por una parte define el punto de partida y, por otra parte, se utiliza como caracter de separación entre los componentes de nombre de ruta. Los tipos de rutas de acceso se explican a continuación:

- **Rutas absolutas.** Representan las rutas que comienzan con la raíz de directorios hasta referenciar el archivo buscado.

Ejemplo: `/home/miguel/fotos/miguel.jpg`

- **Rutas relativas.** Representan las rutas basadas en el principio de directorio actual, es decir, se indica la ruta desde el punto actual del árbol de directorios.

Ejemplo: fotos/miguel.jpg

8.2. Enlaces

Un enlace es un tipo de archivo del sistema UNIX que permite apuntar a otro archivo. Existen dos tipos:

- **Enlaces Simbólicos.** Es un archivo que contiene una ruta a otro archivo de cualquier tipo, esta ruta puede ser relativa o absoluta. Se crea de la siguiente manera:

```
[user@host ~]$ ln -s archivo.txt link_archivo.txt
[user@host ~]$ ls -l link_archivo.txt
lrwxrwxrwx  1 carmen carmen 11 Jul 29 12:09 \
link_archivo.txt -> archivo.txt
```

- **Enlaces Duros.** Es un archivo que apunta al inodo del archivo, por esto no aplica a directorios. Si el archivo fuente es borrado no se rompe el enlace, a diferencia del simbólico. Se crea de la siguiente manera:

```
[user@host ~]$ ln archivo.txt link_archivo.txt
[user@host ~]$ ls -l link_archivo.txt
-rw-rw-r--  2 carmen carmen 0 Jul 29 12:09 link_archivo.txt
```

9. Comandos Básicos

A continuación se seleccionaron los comandos más usados en todos los Sistemas UNIX. Éstos poseen una página del manual (o man).

9.1. Manipulación de Directorios

- **mkdir.** Crea un nuevo directorio cuyo nombre es el pasado por el argumento. Si ya existe un directorio con ese nombre retorna un error.

Modo de uso:

mkdir [opción] directorio...

Las banderas más utilizadas son:

- -p, crea directorios padres como se indique en el argumento. Las rutas pueden ser absolutas y relativas. Por ejemplo:

```
[user@host ~]> mkdir -p dir1/dir2/dir3
[user@host ~]> ls dir1/
dir2
[user@host ~]> ls dir1/dir2/
dir3
```

- **rmdir**. Elimina directorios, los directorios deben estar vacios.

Modo de Uso:

rmdir directorio

- **cd**. Cambia el directorio actual al indicado por el argumento, si no se coloca argumento, el usuario será posicionado en el directorio por defecto de inicio de sesión, usualmente la carpeta de inicio del usuario..

Modo de uso:

cd [directorio]

9.2. Manipulación de Archivos

- **ls**. Lista el contenido del directorio pasado como argumento. Sin argumentos lista el contenido del directorio actual.

Modo de Uso:

ls [opción]... [archivo/dir]...

Las banderas más usadas son:

- -l, lista el contenido usando un formato detallado, donde se pueden ver la permisología, el dueño, el tamaño, fecha de última modificación de los archivos, entre otras cosas.
- -a, lista los archivos ocultos, es decir, cuyo prefijo es punto. Ejemplo: .archivo
- -h, Solo es útil con la bandera -l, e imprime el tamaño de los archivos en unidades conocidas como MB o KB en lugar de cantidad bloques.
- -R, Lista todos los directorios recursivamente.
- -1, Lista los directorios y archivos separados cada uno por una línea.

- **cp**. Copia archivos y directorios. Por defecto no copia directorios.

Modo de Uso:

cp [opción]... fuente destino

cp [opción]... fuente... directorio

Las banderas más comunes son:

- -r -R, Permite la copia recursiva, especial para directorios
- -s, En lugar de copiar realiza enlaces simbólicos.
- -u, Copia solo cuando el archivo fuente es más nuevo que el destino o si no existe.

- **rm**. Elimina archivos y directorios. Por defecto no borra directorios.

Modo de Uso:

rm [opción]... archivo...

Las banderas más usadas son:

- -r -R, Permite la eliminación de directorios.
- -f, Ignora archivos no existentes.
- -i, Pregunta antes de hacer cualquier operación.

- **mv**. Mueve (o renombra) archivos o directorios.

Modo de Uso:

mv [opción]... fuente destino

mv [opción]... fuente... directorio

Las banderas más comunes son:

- -i, Confirma con el usuario antes de sobrescribir.
- -u, Mueve solo cuando el archivo fuente es más nuevo que el destino o si no existe.

- **ln**. Realiza un link entre archivos. Si un nombre de link no es especificado crea uno con el mismo nombre del Target. Por defecto crea un enlace duro.

Modo de Uso:

ln [opción]... target [nombre_link]

ln [opción]... target... Directorio

Las banderas más comunes son:

- -s, Crea un enlace simbólico

9.3. Visualización de Contenido de Archivos

- **cat**. Concatena archivos y los imprime a la Salida Estándar. Si sólo recibe un archivo como argumento muestra su contenido a la salida estándar.

Modo de uso:

cat [opción] [archivo]...

- **more**. Es un visualizador de contenido en formato texto. Se avanza con la tecla espaciadora hacia adelante en el archivo. El archivo se recibe como parámetro.
- **less**. Es como more pero tiene la funcionalidad de poder avanzar y retroceder en el texto. El archivo se recibe como parámetro.
- **head**. Imprime las primeras 10 líneas de un archivo de texto a la salida estándar.

Modo de Uso:

head [opcion]... [archivo]...

Las banderas más utilizadas son:

- -c, Muestra la cantidad de bytes indicada por esta opción.
 - -n, Muestra la cantidad de líneas indicada por esta opción.
- **tail**. Imprime a la salida estándar las últimas líneas de un archivo.

Modo de Uso:

tail [opcion]... [archivo]...

Las banderas más usadas son:

- -c, Muestra la cantidad de bytes indicada por esta opción.
 - -n, Muestra la cantidad de líneas indicada por esta opción.
 - -f, Adjunta a la salida estándar los datos del archivo mientras está siendo escrito, es decir, a medida que crece.
- **wc**. Imprime el número de líneas, palabras y bytes de un archivo.

Modo de Uso:

wc [opcion]... [archivo]...

Las banderas más usadas son:

- -c, Imprime la cuenta de bytes.
 - -m, Imprime la cuenta de caracteres.
 - -l, Imprime la cuenta de líneas.
 - -w, Imprime la cuenta de palabras.
- **nl**. Enumera la cantidad de líneas de un archivo e imprime a la salida estándar.

Modo de Uso:

nl [opción]... [archivo]...

Las banderas más utilizadas son:

- -i, Enumera líneas realizando intervalos de números indicados por el argumento.
- -s, Agrega el string indicado después de cada número.
- -v, Comienza la cuenta desde el número indicado.

9.4. Edición de Archivos

- **vi (o vim)**. Es un editor de texto en modo terminal ampliamente usado por programadores y administradores de sistemas debido a que está presente en cualquier sistema UNIX. Vim es el mismo editor vi pero mejorado. Actualmente en los sistemas linux por defecto vi es vim. Para un aprendizaje básico del editor también puede ejecutar el comando *vimtutor*, el cuál provee un tutorial con ejercicios prácticos sobre el uso de *vi*.

Modo de Uso:

vi [opciones] [archivo ..]

El estilo del editor se basa en varios modos de uso, básicamente el modo de comando y de inserción. Una vez que se abre el editor se esta en modo de comando. Para comenzar a editar el archivo se pasa modo inserción con una tecla específica (i, o, a). Algunos comandos básicos del modo comando se presentan a continuación:

- :help, Muestra la ayuda disponible sobre todos los tópicos de Vi.
- :wq! [archivo], Sale del editor y guarda los cambios
- :wq [archivo], Sale del editor y guarda los cambios.
- :q! (ZQ), Sale del editor sin guardar cambios.

- :x (ZZ), Sale del editor y guarda solo si hubo modificaciones
- /patron, Busca todas las ocurrencias de “patron” en el documento.
- /[RETURN] ?[RETURN], Repite la búsqueda en el documento hacia adelante y hacia atrás, respectivamente.
- h, Se mueve el cursor un espacio a la izquierda.
- j, Se mueve el cursor un espacio hacia abajo.
- k, Se mueve el cursor un espacio hacia arriba.
- l, Se mueve el cursor un espacio a la derecha.
- :#, Se dirige al número de línea indicada por #.
- :\$, Similarmente se dirige a la línea final del archivo.
- dd, Elimina la línea donde el cursor está presente.
- \$, Se posiciona al final de la línea donde se encuentra el cursor.
- 0, Se posiciona al inicio de la línea donde se encuentra el cursor.
- :u, Deshace el último cambio en el documento.
- a, inserta texto un espacio adelante del cursor.

9.5. Impresión

- **lpr**. Imprime archivos a la impresora por defecto. Es usado por el sistema de impresion CUPS.

Modo de Uso:

lpr [opción] archivo

Se usan las siguientes banderas:

- -P, Imprime a la impresora indicado por este argumento.
- -#, Imprime el número de copias indicado, desde 1 hasta 100.
- **lprm**. Elimina un trabajo de la cola de impresión dado un identificador único (un número) dado que cada trabajo tiene este identificador.
- **lpq**. Imprime la cola de impresión del sistema.

9.6. Control de Procesos

- **ps**. Imprime información sobre una selección de procesos activos. Un proceso es una instancia de un programa en ejecución sobre un sistema Linux. Cada procesos se identifica por un número unívoco. Modo de Uso: *ps [options]*

Las banderas más usadas son:

- -A -e, Lista todos los procesos. Muestra identificador único de los procesos (pid), el comando que los inició
- -U. Lista los procesos del usuario indicado.
- -r. Restringe a solo procesos que están corriendo.

Por ejemplo, la ejecución del comando para ver los procesos que estan corriendo en el terminal actual es:

```
[paulus]:/home/usuario>ps
  PID TTY          TIME CMD
 8870 pts/1    00:00:00 bash
 8892 pts/1    00:00:09 kile
 9560 pts/1    00:00:00 ps
```

La salida indica que existen tres procesos, la primera columna es el identificador único del proceso (PID) en el sistema operativo, cada proceso en el sistema tiene uno. La segunda columna indica el terminal donde se invocaron. La tercera columna indica el tiempo que lleva ejecutándose y la última columna se refiere al la linea de comando.

Este comando puede ser ejecutado con distintas banderas que permiten ver con mas detalle la información de los procesos del sistema y de usuario.

- **kill**. Elimina un proceso del sistema. Dependiendo de la señal que se envíe. La señal 9 es la usada para finalizar un proceso, igualmente hay señales para suspender o parar procesos.

Modo de Uso:

```
kill [ -s señal | -p ] [ -a ] [ -- ] pid ...
kill -l [ señal ]
```

Tomando la información del ejemplo anterior si se ejecuta:

```
[paulus]:/home/usuario>kill -9 8892
```

El efecto que tendrá es que terminará abruptamente el proceso con ese PID, ya que se le mando la señal 9 que indica la terminación inmediata del proceso.

- **bg.** Envía al trasfondo (o background) un trabajo activo mediante un identificador del trabajo (que no es lo mismo que pid). Si no ha sido iniciado con el caracter &. Si no se le especifica el parámetro el shell asume que es el último trabajo ejecutado.

Modo de Uso:

```
bg [jobspec]
```

- **fg.** Coloca en el primer plano del shell (o foreground) un trabajo mediante un identificador del trabajo (que no es lo mismo que pid). Si no se le especifica un identificador se asume el último trabajo activo.

Modo de uso:

```
fg [jobspec]
```

10. Archivos, acceso de usuarios y protección

10.1. Tipos de Archivo

La base del sistema de archivos es el archivo. Se diferencia entre distintos tipos:

1. **Archivos Regulares.** Estos pueden contener texto legible o un programa ejecutable (o binarios)
2. **Directorios.** El directorio contiene listas de archivos y de subdirectorios.
3. **Archivos de Dispositivos.** Representan la interfaz a los dispositivos periféricos administrados por el sistema operativo. Cada intento de lectura o escritura en un archivo de dispositivo se transmitirá al dispositivo pertinente.

Para conocer el tipo de un archivo se puede utilizar el comando *file*.

10.2. Control de Acceso y Modos de Protección

Una de las misiones mas importantes de un Sistema Operativo consiste en administrar los datos de una manera ordenada y segura. Para conseguir esa seguridad hay que garantizar que solo pueden acceder a los datos las personas que tengan autorización para verlos o modificarlos. Esa es la razon de que exista control de acceso.

Cada archivo del árbol de archivos de Linux posee lo que se denomina datos de identificacion de archivo. Una forma completa de chequear estos datos es con el comando *ls* y con el parámetro *-l*. Para cada archivo pueden establecerse unos derechos de acceso propios. Cada vez que se accede un archivo, el sistema comprueba si el tipo de acceso que se intenta está permitido.

La salida de *ls -l* muestra la siguiente información:

```
[user@host ~]> ls -l archivo
-rwxrwxr-x NUM usuario grupo # MM DD HH:MM archivo
```

A continuación se explicarán sus campos:

- -. Indica el tipo del archivo. Varía de acuerdo al tipo.

Tabla # 1

-	Archivo Regular
d	Directorios
l	Enlace Simbólico
c	Archivos de Dispositivo de Caracter
b	Archivos de Dispositivo de Bloque

- rwxrwxr-x. Derechos de acceso.

Tabla # 2

Código	Archivo/Directorio
r	Lectura de contenido del archivo / Lectura de archivos del directorio.
w	Modificación del contenido del archivo. / Modificación de archivos en el directorio. Pueden crearse y borrarse archivos.
x	El archivo puede ejecutarse. / Acceso a todos los archivos y subdirectorios contenidos.

- NUM. Número de referencias al archivo, está relacionado con enlaces a archivos explicado previamente.
- usuario. Dueño del archivo.
- grupo. Grupo a que pertenece.
- #. Tamaño del archivo en cantidad de bloques.
- MM DD HH:MM. Fecha de última modificación.
- archivo. Nombre del archivo.

Tabla # 3

Propietario	Grupo	Resto
r w x	r w x	r - x

En cuanto a la modificación de los derechos de acceso, cada archivo es creado con unos determinados derechos de acceso preestablecidos o permisos por defecto. Existen posibilidades y modos de adaptar esos derechos de acceso, para esto se utiliza el comando *chomod*. Este comando puede ejecutarse de dos maneras. La primera denominada simbólica. En la segunda, se trabaja con números octales (cifras en base 8). En esta guía se tratará la primera manera. El comando *chmod* sólo puede utilizarlo el propietario del archivo y el administrador del sistema, que puede hacerlo sobre cualquier archivo.

A este comando se le deben adjuntar dos informaciones distintas, a continuación su sintaxis:

```
chmod tipo_modificacion archivo1 [archivo2 ...]
```

El tipo de modificación se divide en tres aspectos:

- De quién son los derechos que se modificarán. Si se trata de los derechos del usuario (u, user), derechos del grupo (g, group) o del resto (o, others).
- De qué modo se modificarán los derechos. Si se desea añadir nuevos derechos (+) o eliminar algunos de los existentes (-). También es posible modificar los derechos sin tener en cuenta el estado anterior.
- Cuáles derechos de acceso se modificarán: r (lectura), w (escritura) o x (ejecucion)

Por ejemplo:

`chmod g+w archivo`, da permiso de escritura para los miembros del grupo.

`chmod o-r archivo`, resta permiso de lectura al resto.

`chmod ug-w archivo`, resta escritura al dueño y al grupo.

`chmod ug+w,o+r archivo`, concede escritura al dueño y al grupo y lectura al resto.

Sin embargo los derechos de acceso que se guardan para cada archivo no bastan para garantizar su seguridad. Por ello, a cada usuario se le adjudica al identificarse un número único de usuario (UID). La asignación de nombres a los usuarios se lleva a cabo mediante el archivo `/etc/passwd`. Este archivo es un elemento básico de toda la gestión de usuarios. Además, en ese archivo se asigna a cada usuario un número de grupo (GID). De este modo, cada usuario que accede al sistema posee un número de usuario y un número de grupo. Para saber qué número le corresponde, el usuario puede ejecutar el comando *id*.

Evidentemente, es posible cambiar el propietario o el grupo de un archivo o un directorio. Para ello es necesario identificarse como administrador del sistema, es decir, con el nombre del usuario *root* mediante el comando *su - usuario*. Solo en ese caso el comando *chown* permite cambiar el propietario. El comando *chgrp* solo se puede utilizar para hacer accesibles archivos o directorios a otros grupos de los que también se forma parte.

Los comandos *chown* y *chgrp* tienen la siguiente sintaxis:

```
chown usuario archivo1 [archivo2 ...]
chown usuario:grupo archivo1 [archivo2 ...]
chgrp grupo archivo1 [archivo2 ...]
```

11. Expresiones Regulares

Las expresiones regulares se utilizan para hacer búsquedas contextuales y modificaciones sobre textos. Se pueden encontrar en muchos editores de textos avanzados, en programas de análisis gramatical y en muchos lenguajes. Se pueden hacer búsquedas por nombres de archivos o su contenido.

En general podemos asociarlas al shell. Para la definición de criterios de búsqueda se dispone de ciertos caracteres del shell como:

- * Corresponde a cualquier cadena de caracteres de cualquier longitud, incluso vacía.
- ? Corresponde a un solo carácter.
- [] Una sucesión de caracteres, separados por una coma.

- [...] Dentro de los corchetes puede utilizarse un signo de exclamación para convertirlo en excluyente, es decir, todos los caracteres que no aparecen en corchetes.

Linux provee muchos comandos para el uso de expresiones regulares, estas son realmente poderosas y muy usadas por desarrolladores y administradores de sistemas.

11.1. Uso de *grep*

grep. Imprime las líneas que concuerdan con un patrón o expresión regular.

Modo de Uso:

```
grep [opcion] patron [archivo...]
```

```
grep [opcion] [-e patron | -f archivo] [archivo...]
```

Las banderas mas usadas son:

- -a, Procesa un texto binario como si fuera texto.
- -c, Imprime la cantidad de ocurrencias de la correspondencia del patrón.
- -E, Interpreta el patrón como una expresión regular avanzada.
- -e, Procesa el patrón como una expresión regular compleja, equivalente al comando *egrep*
- -H, Imprime el nombre del archivo para cada correspondencia.
- -i, No distingue entre mayúsculas y minúsculas.
- -L, En lugar de imprimir las ocurrencias, muestra el nombre del archivo que no las contiene
- -l, En lugar de imprimir las ocurrencias, muestra el nombre del archivo que las contiene
- -n, Imprime cada ocurrencia en una línea numerada.
- -R -r, Busca correspondencias recursivamente en directorios.
- -v, Muestra todo lo que no concuerda al patrón.

Un pequeño ejemplo sería:

Supongamos que tenemos la siguiente lista de teléfonos de una empresa:

```

Phone Name  ID
...
...
3412   Bob 123
3834  Jonny 333
1248   Kate 634
1423   Tony 567
2567  Peter 435
3567  Alice 535
1548  Kerry 534
...

```

Se trata de una empresa con 500 personas y los datos están almacenados en un archivo ASCII normal. Los registros de personas cuyo teléfono comience con un 1, trabajan en el edificio 1. ¿Quién trabaja en el edificio 1?

Una Expresión Regular puede responder a eso:

```

$ grep '^1' phonenumber.txt
o
$ egrep '^1' phonenumber.txt

```

En palabras normales, esto significa: Busca todas las líneas que comiencen con un 1. El símbolo "^" es el encargado de indicar que sólo se busquen los números 1 que se encuentren al principio de la línea.

Reglas de sintaxis

- **Patrón de un solo símbolo.**

El elemento básico de una expresión regular es el patrón de un solo símbolo. Éste patrón sólo es efectivo cuando este símbolo se puede encontrar exactamente en el texto. Un ejemplo lo podemos encontrar en el número 1 del ejemplo de arriba.

Otro ejemplo para el patrón de un solo símbolo es:

```
$ egrep 'Kerry' phonenumber.txt
```

Este patrón se compone de patrones de un solo símbolo (la letra K, e, etc.)

Varios signos se pueden agrupar en un conjunto. Un conjunto se representa por un par de corchetes (el de abrir y el de cerrar) y una lista de caracteres entre ellos. Un conjunto se considera también como un patrón de un solo símbolo. La búsqueda de este conjunto es efectiva

cuando se encuentre uno y solo uno de los signos del conjunto en el texto. Un ejemplo:

[abc] Un patrón de un solo símbolo con el que se puede encontrar el símbolo a, b o c.

[ab0-9] Un patrón de un solo símbolo en el que se busca una a o una b o un número que se encuentre entre el 0 y el 9 en el código ASCII.

[a-zA-Z0-9_] Esto busca una letra mayúscula o minúscula o un número o el signo -.

En la lista de teléfonos:

```
$ egrep '^1[348]' phonelist.txt
```

Esta expresión busca líneas, que comiencen con 13, 14 ó 18.

La mayoría de símbolos ASCII nos llevan a una búsqueda exitosa cuando se encuentran en el texto, pero también encontramos símbolos en una expresión regular, que tienen un significado especial. Los corchetes hacia la derecha indican el comienzo de la definición de un conjunto. El símbolo '[' en un conjunto es un símbolo especial y representa un rango en el sistema de símbolos ASCII. Para indicar que nos referimos al significado normal del símbolo, se coloca una barra invertida delante '\'. Por ejemplo, en [a-zA-Z0-9_]. En algunos dialectos encontramos que la barra invertida junto con otro símbolo tienen un significado especial. En este caso se obtiene el significado normal retirando la barra invertida.

El punto también es un símbolo importante en una expresión regular. La búsqueda será efectiva, cuando el símbolo comparado sea cualquier símbolo menos el símbolo de nueva línea del código ASCII. Ejemplo:

```
$ grep '^2' phonelist.txt
```

Esta expresión busca líneas que contengan el número 2 en la segunda posición, y cualquier otro símbolo delante.

Los conjuntos se pueden invertir mediante la colocación de "[^" en lugar de "[" en la expresión. En este caso el símbolo "[^" no representa el comienzo de la línea, sino que "[" y "[^" juntos representan el inverso del conjunto.

[0-9] Un patrón de un solo símbolo, la búsqueda será efectiva, si el símbolo en el texto es un número.

[^0-9] Todo lo que no sea un número.

[^abc] Todo lo que no sea una a, b o c. Todo menos el símbolo de línea nueva. Es idéntico a [^\n]. \n es el símbolo para nueva línea.

Para buscar todas las líneas que NO comiencen con un 1, se escribiría:

```
$ egrep '^[^1]' phonelist.txt
```

- **Anclas (anchors).**

Antes hemos visto que "^" representaba el inicio de una línea. Las anclas son símbolos especiales en las expresiones regulares, que no denotan un símbolo, sino una posición.

^ Inicio de una línea

\$ Final de una línea

Para encontrar personas en nuestra lista con un identificador (company-ID) de 567, utilizaremos la expresión:

```
$ egrep '567\$' phonelist.txt
```

Significa: Busca líneas que finalicen con 567.

- **Multiplicadores.**

Los multiplicadores nos indican, cuantas veces ha de aparecer un patrón de un solo símbolo en el texto:

Descripción	grep	egrep
cero o más veces	*	*
una o más veces	\{1,\}	+
cero o una vez	\?	?
de n hasta m veces	\{n,m\}	

Un ejemplo de la lista de teléfonos:

```
.....  
1248 Kate 634  
.....  
1548 Kerry 534  
.....
```

Para encontrar una línea que se componga de un 1, varios números, varios espacios y la letra K se escribirá:

```
grep '^1[0-9]\{1,\} \{1,\}K' phonelist.txt
```

o se utiliza * y se repite [0-9] y el espacio:

```
grep '^1[0-9][0-9]* *K' phonelist.txt
```

o

```
egrep '^1[0-9]+ +K' phonelist.txt
```

Es importante saber que un multiplicador es ávido. Esto significa que el primer multiplicador se extenderá lo máximo que pueda hacia la derecha.

Esto no es muy importante en grep, pero es importante en el momento de editar textos.

La expresión `^1.*4` se extendería por toda la línea 1548 Kerry 534 de principio a fin.

La expresión no se extiende sólo hasta la zona corta, o sea, 154, sino hasta el máximo que se pueda.

■ Paréntesis como memoria.

Los paréntesis como memoria no influyen en el tipo o símbolos por los que se buscará. Sirven para que los fragmentos de texto se almacenen y pueda accederse a ellos a través de variables.

El primer paréntesis será referenciado como variable 1, el segundo como variable 2, etc.

Nombre del Programa	Sintaxis del Paréntesis	Sintaxis de las variables
grep	\(\)	\1
egrep	()	\1

Un ejemplo: La expresión `[a-z] [a-z]` busca dos letras minúsculas.

Ahora se pueden utilizar estas variables, para buscar patrones de texto como `?otto?`:

```
$ egrep '([a-z])([a-z])\2\1'
```

La variable `\1` contiene la letra o y `\2` la letra t. La expresión serviría también para `anna`, pero no para `yxyx`.

Los paréntesis no se utilizan normalmente para la búsqueda de nombres como `otto` y `anna`, si no para editar o para buscar y sustituir.

12. Redireccionamiento de Entrada, Salida y Error Estándar

El redireccionamiento de entrada y salida es una sencilla y a la vez sorprendente propiedad del shell. Todos los comandos utilizan los canales de entrada y salida para reproducir o leer sus datos. El canal de entrada, normalmente utilizado para la lectura, está vinculado al teclado. El canal de salida estándar está vinculado a la pantalla. Linux ejecuta un canal de entrada y salida distinto para cada usuario. Por otro lado el error estándar es donde el shell redirecciona los errores y el canal que utiliza por defecto es la pantalla.

El shell puede manipular brevemente estos canales de entrada y salida para un comando introducido por el usuario, de forma que dejen de estar vinculados al teclado o a la pantalla y se vinculen a un archivo.

Entonces los comandos leen sus datos directamente del archivo o escriben sus datos en el archivo. Usualmente escriben en la salida estándar.

El redireccionamiento está representado por los símbolos `>`, `<`, el primero se asocia al redireccionamiento de salida y el segundo al de la entrada.

Un ejemplo de redireccionamiento de salida es:

```
[user@host dir]> ls -l > lista
[user@host dir]> more lista
total 0
-rw-rw-r-- 1 carmen carmen 0 Jul 29 14:59 archivo1
-rw-rw-r-- 1 carmen carmen 0 Jul 29 14:59 archivo2
-rw-rw-r-- 1 carmen carmen 0 Jul 29 15:00 lista
```

Un ejemplo de redireccionamiento de entrada es:

```
[user@host dir]> wc < archivo1
3 3 21
```

También se usan dos variaciones de estos símbolos `>>`, `<<`, ya que no tiene sentido sobrescribir el contenido de un archivo anterior se pueden utilizar para adjuntar texto en un archivo. Por ejemplo:

```
[user@host ~]> more archivo1
linea1
[user@host ~]> echo "linea2" >> archivo1
[user@host ~]> more archivo1
linea1
linea2
```

13. Pipes

Para realizar tareas en el shell, en general Unix provee algunas herramientas para manipulación y unión de salidas y entradas de comandos. Por ejemplo, sería muy interesante poder unir la salida de un programa con la entrada de otro y construir una cadena de órdenes. Como hacer un `cat` de un archivo y unir la salida con el comando `sort` luego redireccionar al archivo donde queremos guardar la salida. Para esto se usa el carácter `|`, este carácter permite crear un pipe que es simplemente unir la salida de un comando con la entrada de otro. A continuación un ejemplo:

```
[paulus]:/home/carmen>cat lista_compra
lechuga
papas
harina
leche
zanahorias
naranjada
avena
[paulus]:/home/carmen>cat lista_compra |sort -n
avena
harina
leche
lechuga
naranjada
papas
zanahorias
```

14. Variables de Ambiente

Las Variables de Ambiente son un conjunto dinámico de valores que pueden afectar el comportamiento de los procesos activos.

Estas pueden ser globales o locales. En el primer caso, están disponibles en todos los shells activos del sistema y en el segundo caso, solo concierne al shell actual.

Las variables de ambiente son manejadas por el shell. La diferencia entre variables de ambiente y variables regulares del shell es que una variable del shell es local a esa instancia particular del shell, mientras que las de ambiente son heredadas por cualquier programa que se inicie, incluyendo otro shell. En la mayoría de los shells de Unix, cada proceso tiene su propio conjunto de

variables de ambiente que son copias de las variables de ambiente del proceso padre.

Estas variables se escriben comúnmente en letras mayúsculas y se le asigna su valor de la siguiente manera:

```
$ var=value
```

Algunos ejemplos de variables de ambiente reservadas incluye:

1. PATH. Indica los directorios separados por punto y coma, donde buscará el shell por comandos que el usuario escribe. La búsqueda se realiza de izquierda a derecha.
2. HOME. Indica donde se encuentra localizado en el sistema de archivos, el directorio personal de un usuario.
3. LOGNAME/USER. El nombre de usuario.
4. PRINTER. Impresora usada por defecto vía *lpr*.
5. SHELL. El shell usado por el usuario.
6. HOSTNAME. El nombre del host.
7. PS1. El prompt por defecto para *bash*.
8. PS2. El prompt secundario para *bash*.
9. PWD. Contiene la ruta absoluta del directorio actual.

Los scripts del Shell (Shellscripts) usan variables de ambiente para guardar valores temporales para referenciarlos posteriormente.

Las variables pueden ser usadas en scripts y en la línea de comando. Ellas son referenciadas colocando un símbolo especial \$ delante. Para mostrar el contenido de la variable PATH se utiliza:

```
$ echo $PATH.
```

```
$ echo ${PATH}.
```

Para mostrar todas las variables de ambiente definidas para el usuario se utiliza el comando:

```
$ env
```

```
$ printenv
```

```
$ set
```

El comando *set* mostrará tanto las variables locales como globales a diferencia de *env* y *printenv*.

Para modificar el valor de una variable de ambiente durante la ejecución de una instancia del shell se utilizan los siguiente comandos:

\$ var=value
\$ export VAR=valor
\$ setenv VAR valor

15. Comandos para compresión de Archivos

Linux provee muchos comandos de compresión de archivos, los más usados son:

- zip. Es una combinación de los comandos *compress* y *tar*. Le da la extensión *.zip*. Su sintaxis es la siguiente:

```
zip [opciones] archivozip [archivo1...]
```

Algunas de sus opciones son:

- -b ruta, Usa la ruta especificada para colocar temporalmente el archivo zip, cuando finalice lo copia al directorio donde se realizó la operación.
- -c, Agrega una línea de comentario por cada archivo, el usuario introduce por la entrada estándar cada comentario.
- -d, Elimina entradas de un archivo zip.
- -e, Encripta el contenido del archivo zip con una contraseña que introduce el usuario por la entrada estándar.
- -f, Reemplaza un entrada existente en el archivo zip, solo si ha sido modificada más recientemente que la versión contenida en el archivo zip.
- -F, Repara el archivo zip.
- -g, Añade al archivo zip especificado en lugar de crear uno nuevo.
- -i archivos, Incluye solo los archivos especificados.
- -r, Realiza la compresión recursivamente a través de los directorios.
- -T, Prueba la integridad del archivo zip creado.
- -x, Excluye explícitamente los archivos indicados.
- -#, Regula la velocidad de compresión usando el dígito indicado, donde -0 indica sin compresión.
- -@, Toma la lista de archivos de la entrada estándar.

- unzip. Lista, prueba y descomprime los archivos que comprimió *zip*. Su sintaxis es la siguiente:

```
unzip [opciones] archivo[.zip] [archivo(s) ...] \  
[-x xarchivo(s) ...] [-d exdir]
```

Las opciones más resaltantes son:

- -x xarchivo(s). Recibe una lista de miembros del archivo zip a ser excluidos del procesamiento.
 - -d exdir. Se indica un directorio opcional donde se extraerán los archivos.
 - -f, Refresca los archivos existentes, es decir, solo aquellos archivos que se extraigan que existan en disco y que sean más nuevos que los que están en disco.
 - -l, Lista los archivos en formato corto de un archivo zip.
 - -t, Prueba los archivos zip.
 - -z, Muestra solo los comentarios de las entradas.
 - -u, Actualiza los archivos existentes y crea los que se requieran.
- gzip. Reduce el tamaño de los archivos usando un algoritmo más eficaz. Le añade la extensión *.gz*. Su sintaxis es:

```
gzip [opciones] [-S sufijo] [nombre ...]
```

Las opciones más importantes son:

- -c, Escribe a la salida estándar, mantiene los archivos originales sin cambios.
- -d, Descomprime.
- -f, Obliga la compresión o descompresión incluso si el archivo tiene múltiples enlaces o si existe el archivo que quiere ser creado.
- -l, Por cada archivo comprimido lista la siguiente información: tamaño del archivo comprimido, tamaño del archivo sin compresión, la tasa de compresión y el nombre del archivo no comprimido.
- -r, Realiza la compresión o descompresión recursivamente en los directorios.
- -S, .suf, Usa el sufijo *.suf* en lugar de *.gz*.

- -t, Realiza pruebas de integridad.
 - -# -fast -best, Regula la velocidad de compresión.
- gunzip. Este descomprime archivos creados por *gzip*, *zip*, *compress*, y *pack*. Utiliza las opciones de *gzip* y su sintaxis es la siguiente:

```
gunzip [opciones] [-S sufijo] [nombre ...]
```

- bzip2. Comprime archivos usando un algoritmo más eficaz que los anteriores. Este espera una lista de nombres, donde cada archivo es reemplazado por una versión comprimida de sí mismo con la extensión *.bz2*. Cada archivo conserva sus atributos para que se puedan restaurar al descomprimirlos. Su sintaxis es:

```
bzip2 [opciones] [archivos...]
```

Las opciones más usadas son:

- -c, Comprime o descomprime a la salida estándar.
 - -d, Fuerza la descompresión.
 - -t, verifica la integridad de los archivo(s) especificados pero no lo descomprime.
 - -f, Fuerza la sobrescritura de los archivos de salida.
 - -k, Mantiene los archivos de entrada durante la compresión y descompresión.
 - -1 (o -fast) hasta -9 (or -best), Establece el nivel de compresión.
- bunzip2. Descomprime todos los archivos especificados, ignorando aquellos que no fueron comprimidos con *bzip2*. Las opciones que utiliza son las explicadas previamente y su sintaxis es la siguiente:

```
bunzip2 [opciones] [archivos ...]
```

Adicionalmente se provee una herramienta para recuperar datos dañados durante la compresión o descompresión, esta se llama *bzip2recover*

16. Respaldo y Recuperación de Archivos: *tar*

El comando *tar* permite el empaquetado de información en un archivo. Este archivo es conocido como un *tarfile*.

Sintaxis: *tar [opciones] archivo1 archivo2 directorio1 directorio2]*

El uso de las opciones, a diferencia de la gran mayoría de los comandos UNIX, no necesita que se preceda con el símbolo “-”. Las opciones más usadas son:

- *c* Crea un archivo *tar*.
- *x* Extrae información de un archivo *tar*.
- *t* Lista el contenido de un archivo *tar*.
- *v* Muestra información adicional de lo que está ejecutando el comando *tar*.
- *p* Conserva la permisología de los archivos.
- *f* Especifica el archivo de origen o destino.
- *j* Indica que el archivo de origen o destino está comprimido o será comprimido usando el comando *bzip2* o *bunzip2*.
- *z* Indica que el archivo de origen o destino está comprimido o será comprimido usando el comando *gzip* o *gunzip*.
- *Z* Indica que el archivo de origen o destino está comprimido o será comprimido usando el comando *compress* o *uncompress*.

Ejemplo:

- *tar cf* archivo.tar directorio
- *tar cvf* arvhivo.tar directorio
- *tar cvpf* archivo.tar directorio
- *tar tf* archivo.tar
- *tar cfz* archivo.tgz directorio
- *tar tvfz* archivo.tgz
- *tar cpvfj* archivo.tar.bz2 directorio
- *tar xvfj* archivo.tar.bz2

17. Secure Shell: SSH

El SSH (el comando `ssh`) es un conjunto de estándares y un protocolo de red asociado que permite establecer un canal de comunicación seguro entre una máquina remota y una máquina local. Provee un shell remoto para ejecutar comandos en la máquina remota. La máquina remota contiene al proceso que provee el servicio llamado `sshd` y la máquina local tiene al cliente llamado `ssh`. SSH es una versión segura del comando `rsh` (*Remote Shel*).

Por ejemplo si ud. se encuentra en la máquina `taraza ldc.usb.ve` y quiere conectarse a la máquina `elrood ldc.usb.ve` ejecuta:

```
[taraza]:/home/usuario>hostname
taraza
[taraza]:/home/usuario>ssh elrood ldc.usb.ve -l usuario
usuario@elrood ldc.usb.ve's password: <Se introduce la contraseña>
Last login: Sat May 6 17:07:30 2006 from maquina
[elrood]:/home/usuario>hostname
elrood
```

17.1. Características

- Las contraseñas no se transmitan claramente a través de la red.
- Uso de autenticación fuerte en los sistemas a conectar, pero no sólo basado en el nombre o dirección IP que están sujetos a spoofing. Esta basada en algoritmos de cifrado usando claves públicas (tanto para sistemas como usuarios).
- Ejecución de comandos remotos con total seguridad.
- Protección de las transferencias de archivos y directorios.
- Seguridad de las sesiones X11 que son muy vulnerables.
- Reemplazo de los comandos: `ssh` en vez de `rsh` y `rlogin`, `scp` con `rcp`, `sftp` con `ftp`;
- Capacidad de redirigir el flujo de datos TCP en un "tunel" por sesión y en particular en sesiones X11 donde se puede hacer automáticamente.
- Cifrado del tunel, y cuando sea necesario y especificado, comprimirlo.

18. Programacion con el Shell “Shells scripting”

Una de las grandes ventajas de Unix, es que esta hecho de pequeñas herramientas individuales, mediante el uso de pequeños bloques como *cat* y *grep* que se pueden usar rápidamente desde el prompt o shell. Usando pipes, redirecciones, filtros y otras cosas, se pueden manipular todas estas utilidades para realizar un gran número de cosas. La programación con el Shell permite tomar los mismos comandos que se escriben en el prompt y colocarlos en un archivo que se pueda ejecutar con solo escribir su nombre.

Debido a que los usuarios de UNIX usan el shell todos los días, no necesitan aprender un nuevo lenguaje para llevar a cabo el desarrollo de programas, solo seguir algunas pautas o técnicas que incluso se podrían usar directamente desde el prompt del terminal. Esta sección se encarga de explicar las técnicas necesarias para crear nuevos programas en el shell de UNIX.

En el caso más simple, un shell script no es más nada que un grupo de comandos del sistema guardados en un archivo. Por lo tanto, esto disminuye el esfuerzo de reescribir una secuencia de comandos en particular, cada vez que se necesita hacer un trabajo.

Ejemplo 1: Un script para limpiar logs del sistema.

```
# Script que limpia archivos de logs en el directorio /var/log
cd /var/log
cat /dev/null > messages
cat /dev/null > wtmp
echo ‘‘Logs limpios.’’
```

Coloque la información anterior en un archivo llamado “script.sh” y luego cambie los permisos, activando los permisos de ejecución, ejecutando el siguiente comando:

```
chmod 500 script.sh
o también
chmod u+rx script.sh
```

Al especificar los permisos se debe tener en cuenta que el archivo DEBE tener permisos de lectura y ejecución, los primeros debido a que el sistema necesita leer el interior del archivo, y los permisos de ejecución son necesarios para ejecutar el script propiamente. Una vez aplicados los permisos indicados, se ejecuta el script escribiendo la instrucción de la siguiente manera:

```
./script.sh
```

El ejemplo anterior, no tiene nada fuera de lo común. Representa un archivo que contiene un grupo de instrucciones una después de la otra, y luego de ejecutarse, los logs del sistema quedan vacíos. Este script muy sencillo, ha

disminuido la tarea de realizar tres instrucciones seguidas, cada vez que se requiera limpiar estos archivos.

La primera línea del script, es denominada “sha-bang” (!#) y le dice al sistema que el archivo es un conjunto de instrucciones que pertenecen al interpretador de comando indicado. Los caracteres #! son los primeros dos bytes del archivo e indican el “magic number”, un marcador especial que se usa para clasificar el tipo de archivo, en este caso, el archivo se trata de un shell script ejecutable (para mayor información del “magic number” ejecute *man magic*). Inmediatamente a los caracteres “sha-bang” se encuentra una ruta completa que indica el intérprete usado para ejecutar el script, puede ser un shell, un lenguaje de programación o una utilidad. Este comando interpreta y ejecuta los comandos indicados en el script, del principio al final ignorando los comentarios.

Algunos intérpretes usados en shell scripts son los siguientes:

- #!/bin/sh
- #!/bin/bash
- #!/usr/bin/perl
- #!/usr/bin/tcl
- #!/bin/sed -f
- #!/usr/awk -f

Cada uno de las cabeceras indicadas arriba, llama a un intérprete de comandos distintos. En particular el más usado es el #!/bin/sh, que es el shell de inicio que viene por defecto en casi todas las variantes de UNIX. El uso de este intérprete garantiza la portabilidad del programa a otros sistemas UNIX.

Notese que la ruta dada en el “sha-bang” debe ser correcta, de lo contrario, el script indicará un error de “Command not found”. La línea de “sha-bang” puede ser omitida solamente si el archivo consta comandos genéricos del sistema y no usa directivas internas del shell. En la segunda versión del script es necesaria la línea del “sha-bang” debido a las asignaciones realizadas en el cuerpo del programa.

Una vez que se declara la línea de “sha-bang”, el script procede a inicializar un grupo de variables. La cláusula *if* se encarga de ejecutar **test** dentro del programa para definir si se sigue la ejecución del mismo. En particular se hacen dos tests importante, el primero que determina si el usuario que ejecuta el script es el usuario ROOT, y el tercer condicional que determina si se esta

trabajando en el directorio correcto. La segunda comparación determina si se ha ejecutado el script indicando argumentos adicionales en la línea de comando.

18.1. Declaración de variables

En todo lenguaje de programación existe el concepto de variables. Una variable es un buffer o espacio temporal de memoria que puede ser conservado en la ejecución del programa para guardar información que está cambiando constantemente o para modificar el comportamiento de un programa en base a su valor. En BASH las variables no tienen tipo, esto significa, que no se debe preocupar si la variable debe declararse como una variable tipo entero para guardar información de números o tipo "string" para guardar texto claro, esto en BASH puede hacerse indistintamente en ambos casos. Para declarar una variable basta escribir la expresión:

VARIABLE=valor

No se debe colocar espacio alguno entre el símbolo "-". Para hacer referencia luego a el valor de la variable se usa la instrucción con la expresión \$VARIABLE.

Ejemplos:

- VAR=25
- hola="hola"
- hola2=\$hola
- letras="a b c d f"

Luego para mostrar el valor de las variables se usa:

echo \$VARIABLE

Cuando se usa la instrucción *echo* para determinar el valor de una variable se debe tomar en cuenta: La ejecución de la instrucción con las comillas dobles (") ejecuta la sustitución de las variables dentro de la instrucción si la hubieran. Si se usan comillas simples (') el resultado es que los caracteres especiales del shell se interpretan literalmente y sin significado especial.

Ejemplo 1.

```
#!/bin/bash
hello="A B C D"
echo $hello # A B C D
echo "$hello" # A B C D
```

```
# As you see, echo $hello and echo "$hello" give different results.
# Quoting a variable preserves whitespace.
```

```
echo
```

```
echo '$hello' # $hello
```

La instrucción *unset VARIABLE* elimina el valor de la variable y la convierte en **null**. Notese que el valor que se le da a la variable NO es “” o “cero”, directamente toma el valor de “null” y haciendo comparaciones en un script luego de haber ejecutado la instrucción *unset* es un error muy común. Notese también que el nombre de la variable en la instrucción no está acompañada por el caracter “\$”.

Una particularidad que tiene la asignación en shell scripts, que es posible asignar el resultado de un comando a una variable, esto usando las comillas llamadas “backquotes” ‘`’`.

Ejemplo 2

```
#!/bin/bash
a='ls -la'
echo $a
echo '$a'
echo '$a'
exit 0
```

Existe un grupo de variables especiales que pueden ser usadas en el programa y son reservadas por el shell:

- \$0 Indica el nombre del programa
- \$1,\$2, \$3, ... , \$9, \$10, \$11, ... Indican los argumentos pasados al script, cada uno indica la posición en la cual fueron enviados, \$1 se refiere al primer argumento, \$2 se refiere al segundo, así sucesivamente. Nótese que después del noveno argumento, la sintaxis cambia, pero el uso es el mismo.
- \$* Indica todos los argumentos pasados como parámetros.
- \$# Indica el número de parámetros pasados al script.
- \$? Indica el status devuelto por la última llamada al sistema o la ejecución de un script.

Ejemplo 3

```
#!/bin/bash
salida=0
echo ‘Nombre del programa’
echo $0
echo ‘Primer parámetro’
echo $1
echo ‘Segundo parámetro’
echo $2
echo ‘Lista de parámetros’
echo $*
echo ‘Número de parámetros’
echo $#
ls -la
echo $?
ls -la fantasma.txt
echo $?
echo fin del programa
exit $salida
```

18.2. Test y Comparaciones

Las comparaciones nos permiten evaluar condiciones que afectan el flujo normal del programa, con la intención de crear tareas más elaboradas de programación. La construcción que permite esto en “bash” es la siguiente:

```
if [ condicion1 ]
then
    comando1
    comando2
    comando3
elif [ condicion2 ]
# Es lo mismo que else if
then
    comando4
    comando5
else
    Comandos por defecto
fi
```

La condición1 se ejecuta en caso de que el valor de salida que contienen los elementos del test sea igual a cero (0), y la condición se vuelve falsa en cualquier otro caso. Si no ocurriese que sea verdadero, se evalúa la condición2 en caso de que exista, y por último, y también opcional para el programador se ejecutan las instrucciones en el bloque delimitado por la cláusula “else” siempre y cuando todas las demás condiciones sean falsas.

Los operadores que pueden ser utilizados en las cláusulas demarcadas por las condiciones son los siguientes:

- -e Evalúa que un archivo exista.
- -f Evalúa que el archivo sea regular, es decir, no es un directorio ni un archivo de dispositivos.
- -s Evalúa que el archivo no sea de longitud cero.
- -d Evalúa si el archivo es un directorio.
- -c El archivo es de caracteres (modem, teclado, mouse,etc)
- -b El archivo es de bloques (floppy, cdrom, etc)
- -L El archivo es un link simbólico.
- -r El archivo tiene permisos de lectura.
- -w El archivo tiene permisos de escritura.
- -x El archivo tiene permisos de ejecución.
- archivo1 -nt archivo2 Es cierto si el archivo1 es más nuevo que el archivo2.
- archivo1 -ot archivo2 Es cierto si el archivo1 es más viejo que el archivo2.
- archivo1 -ef archivo2 Es cierto si ambos archivos se refieren al mismo dispositivo o al mismo inodo.
- -z texto Indica si la longitud del texto es cero (0).
- -n texto Indica si la longitud del texto NO es cero.
- texto1 = texto2 Indica si el texto1 y el texto2 son iguales.
- texto1 != texto2 Indica si el texto1 y el texto2 son diferentes.

- `texto1 > texto2` Indica si el `texto1` es lexicograficamente mayor que `texto2`.
- `texto1 < texto2` Indica si el `texto1` es lexicograficamente mayor que `texto2`.

Ejemplo 4

```
#!/bin/bash
#
# Script para pruebas de tests
#
#

FANTASMA=/etc/noexiste
REGULAR=/etc/passwd
DIRECTORIO=/home
TEXT01=""
TEXT02="Texto no vacio"
TEXT03="Texto no vacio"

NUM1=10
NUM2=5
NUM3=8

if [ -e $FANTASMA ]
then
    echo El archivo $FANTASMA SI existe
else
    echo El archivo $FANTASMA NO existe
fi

if [ $NUM1 -lt $NUM2 ]
then
    echo $NUM1 es menor que $NUM2
else
    echo $NUM2 es menor que $NUM1
fi

if [ "$TEXT03" = "$TEXT02" ]
then
```

```

        echo Los textos son iguales
else
        echo Los textos no son iguales
fi
exit 0

```

18.3. Ciclos

Un ciclo, es un grupo de instrucciones que se ejecutan mientras cierta condición es verdadera.

La construcción básica para construir un ciclo del tipo “for” difiere de las usadas en los lenguajes típicos de programación:

```

for arg in [list]
do
    instruccion1
    instruccion2
done

```

En este caso “arg” se refiere a una variable y [list] se refiere a una lista de valores que puede ser explícita o como el resultado de la salida de un comando.

Ejemplo 5

```

#!/bin/bash
# Un ciclo for con [list] generada por sustitución de comando.

NUMBERS="9 7 3 8 37.53"

for number in `echo $NUMBERS` # for number in 9 7 3 8 37.53
do
    echo -n "$number "
done

echo
exit 0

```

Ejemplo 6

```

#!/bin/bash
# Script que cuenta los números del 1 al 10

```

```

echo

# Sintaxis estandard.

for a in 1 2 3 4 5 6 7 8 9 10
do
    echo -n "$a "
done

```

Para la construcción del ciclo del tipo “While” la sintaxis es la siguiente:

```

while [condition]
do
    Instrucciones...
done

```

Ejemplo 7

```

#!/bin/bash

echo

while [ "$var1" != "exit" ]      # while test "$var1" != "end"
do
    echo "Ingrese variable #${i} (exit para salir) "
    read var1                    # No se usa 'read $var1'.
    echo "variable #${i} = $var1"
    echo
    i=$((i+1));
done

exit 0

```

Las instrucciones “break” y “continue” dentro de un ciclo tiene el mismo significado que sus análogos en los lenguajes de programación convencionales, es decir, el uso de “break” causa una interrupción en la ejecución del ciclo mientras que la instrucción “continue” se encarga de hacer un salto inmediato a la proxima iteración del bloque de instrucciones del ciclo al cual pertenece.

19. Ejercicios

1. >Qué salida muestran los siguientes comandos? Consulte las páginas del manual (man) y explique cada una.

1. `man read`
2. `man 2 read`
3. `man -S 2 read`
4. `man -W read`

2. >Qué significan los archivos `.` y `..` en un directorio.

3. ¿Qué acción realiza el comando `file *`?

4. Qué ocurre si se ejecutan los siguientes comandos:

1. `mkdir -v {dir1,dir2}`
2. `mkdir -p dir1/{dir2,dir3}`
3. `mkdir -m 700 dir4`
4. `mkdir -m 777 dir1/dir2/{dir3,dir4}`
5. `mkdir dir1/{dir2,dir3}/dir4`
6. `mkdir -vp dir1/{dir2,dir3,dir4}{1,2,3}/dir{4,5,6}`

5. Si ejecuta el comando `cd` sin parámetros, ¿En qué directorio se encuentra ud.? ¿Por qué sucede esto?

6. Asimismo si ejecuta el comando `cd ..`, ¿En qué directorio se encuentra ud.? ¿Por qué sucede esto?

7. Si ejecuta el comando `cd ~`, ¿En qué directorio se encuentra ud.? ¿Por qué sucede esto?

8. ¿Qué comandos dará la información del tamaño de cada archivo en el directorio actual?

1. `ls -l`
2. `ls -la`
3. `du .`
4. `ls -a`
5. `ls`

9. Indique el resultado de la ejecución de los siguientes comandos:

1. `ls -R`
2. `ls -ah`
3. `cp -R dir1 dir2`
4. `cp . . .`
5. `rm dir1`

10. ¿Qué acción realizan los siguientes comandos?

1. `cp --preserve *.c archives`
2. `cp /home/jenny/memo .`
3. `cp -l max dir`
4. `cp memo letter /home/jenny`

11. ¿Cuál de los siguientes comandos usaría para mostrar la salida una página a la vez de un archivo?

1. `more`
2. `less`
3. `sed`
4. `pause`
5. `grep`

12. Indique los comandos para:

1. Las primeras 100 líneas de un archivo.
2. Las últimas 100 líneas de un archivo.
3. Indicar cuántas líneas tiene un archivo.

13. ¿Qué ocurre si ejecuto la siguiente secuencia de comandos? ¿Por qué ocurre esto?

```
$ vi archivo &  
$ fg
```

14. ¿Que sucede si ejecuta la siguiente secuencia de comandos?

```
$ find / -name wp -print > wp.list1  
CTRL-Z  
$ bg
```

15. ¿Cuál es el comando básico para imprimir archivos? ¿Cuál es el comando para ver el status de la impresora?
16. ¿Qué significan los siguientes permisos? ¿Estos permisos pertenecen a directorios o archivos?
1. `drwxr-xr-x`
 2. `-r-wr--r--`
 3. `-rw-r--r--`
17. ¿Qué sucede si a un directorio tiene los siguientes permisos:
1. `d-----`
 2. `d--w-----`
 3. `d-w-----`
 4. `d-wx-----`
 5. `dr-----`
 6. `dr-x-----`
 7. `drw-----`
 8. `drwx-----`
18. Si desea conseguir todos los archivos de tres letras que terminen por la letra `y` en el directorio actual. ¿Qué comando usaría?
1. `ls *y`
 2. `ls *y*`
 3. `ls ??y`
 4. `ls ??y*`
19. ¿Qué criterio de búsqueda es el mejor para encontrar las líneas que contengan la palabra `MERCURY` en el archivo “cliente” ?
1. `grep cliente MERCURY`
 2. `find cliente MERCURY`
 3. `sed cliente MERCURY`
 4. `search cliente MERCURY`
20. ¿Qué búsquedas realizan los siguientes comandos?

1. `grep cicl Vehiculos`
2. `grep "[Cc]icl" Vehiculos`
3. `grep "^o" Vehiculos`
4. `grep -i "cicl" Vehiculos`
5. `grep -v "cicl" Vehiculos`
6. `grep -vi "cicl" Vehiculos`
7. `grep -n "l$" Vehiculos`
8. `grep -vc "cicl" Vehiculos`

21. Ejecute los siguientes comandos como usuario sin privilegios (no root). Determine la entrada, salida y error estándar por cada comando.

1. `cat noexiste`
2. `file /sbin/ifconfig`
3. `grep root /etc/passwd /etc/nofiles > grepresults`
4. `/etc/init.d/sshd start > /var/tmp/output`
5. `/etc/init.d/crond start > /var/tmp/output 2>&1`

Nota: Ahora revise sus resultados ejecutando los comandos nuevamente pero redireccionando la salida estándar al archivo "salida" y el error estándar al archivo "error".

22. Use el comando *tail* para mostrar las últimas 10 líneas del archivo `/etc/passwd` y redireccione la salida estándar al archivo "prueba".