Sandra Liliana Allende
Fabian Alejandro Gibellini
Cecilia Beatriz Sánchez
Monica Mariel Serna

SISTEMA OPERATIVO LINUX

Teoría y Práctica

2da. Edición

70 años UON·UTN 1948 2018

CiN REUN
Red de Editoriales de Universidades Nacionales de la Argentina

L U A
Libro Universitario Argentino

edUTecNe

SISTEMA OPERATIVO LINUX

Teoría y Práctica

2da. Edición

Sistemas operativos : Linux teoría y práctica / Sandra Liliana Allende, Fabian Alejandro Gibellini, Cecilia Beatriz Sánchez, Monica Mariel Serna - 2a ed ampliada.
Ciudad Autónoma de Buenos Aires
edUTecNe, 2019.
Libro digital, PDF

Archivo Digital: descarga y online
ISBN 978-987-4998-13-2

1. Sistema Operativo. 2. Córdoba . 3. Investigación Teórica.
CDD 005.432

Diseño de interior y tapa: Fernando Cejas, Carlos Busqued

Queda hecho el depósito que marca la Ley Nº 11.723

Sarmiento 440, Piso 6 (C1041AAJ)
Buenos Aires, República Argentina
Publicado Argentina – Published in Argentina

Agradecimientos

A nuestras familias por su comprensión, a nuestros colegas de la cátedra de Sistemas Operativos de la UTN-FRC por sus aportes y sugerencias, y a nuestros alumnos que con sus inquietudes y necesidades dieron origen a este trabajo.

Prólogo

Esta publicación tiene por finalidad apoyar en su aprendizaje a nuestros alumnos de la Catedra de Sistemas Operativos de la carrera de Ingeniería de Sistemas de Información y se caracteriza por ser una guía Teórico-Práctica que contiene los conceptos teóricos propios de Linux y a su vez relaciona y refuerza los conceptos de los Sistemas Operativos en general.

Al finalizar la primera edición nos comprometimos a realizar una segunda agregando temas que consideramos que faltaban tratar y que no pudimos incluir en ese momento. Es por ello que, en esta nueva obra, tratamos de completar y actualizar todos los conceptos abordados en la primera etapa, y también incluir nuevos temas que consideramos necesarios tendiendo a una constante mejora.

Hemos dedicado muchas horas de trabajo e investigación cuya meta era la de lograr un material didáctico lo más completo posible y en un lenguaje entendible para el alumno. Es por ello, que la labor terminada, nos ha dejado la grata sensación de haber cumplido nuestros objetivos.

No queremos dejar pasar la oportunidad de reforzar la siguiente idea: Aprender implica esfuerzo. Esta es la idea que se ha ido perdiendo. Que aprender es un trabajo, una tarea nada sencilla que demanda sacrificios. Que aprender supone que cada persona emprenda un esfuerzo individual destinado a modificarse a sí misma, por lo general con ayuda de otros. Allí interviene el docente, que con su conocimiento y experiencia guía al alumno a esa exploración y, sobre todo, lo motiva a emprenderla.

Afirma un proverbio chino: "Los maestros abren la puerta, tú debes entrar por ti mismo".

Introducción a la obra

Esta segunda edición, al igual que la primera, sigue siendo una obra con fundamentos de cómo el Sistema Operativo Linux gestiona los recursos de un sistema informático, y aborda conceptos relacionados con la administración de Linux.

Cada tema fue tratado conceptualmente y en forma práctica, y desarrollado con un nivel homogéneo de profundidad, para que el estudiante cuente con las bases necesarias para abordar el fascinante mundo de Linux y le sea posible la investigación posterior de los temas que sean de su interés.

Su contenido incluye fundamentos teóricos, acompañados de actividades prácticas de laboratorio auto-contenidas, es decir, actividades resueltas y explicadas paso a paso, para que el alumno pueda desarrollar, si fuese necesario, fuera del aula del laboratorio. Por este motivo se hace deseable disponer de al menos una PC para seguir las actividades que se van planteando.

La organización de su contenido está dividida en 13 capítulos o, según nosotros, grandes temas. Al final de cada capítulo hay actividades para reforzar la teoría y/o actividades prácticas de laboratorio. Estas actividades integran los conceptos vistos en temas anteriores, están resueltas y acompañadas de una breve explicación conceptual.

Además de las actividades finales de cada capítulo, se anexa una guía de ejercicios de práctica libre, también resueltos, cuyo fin es reforzar los conceptos desarrollados, estas actividades se encuentran bajo el titulo de Prácticos Integradores.

ÍNDICE DE CONTENIDOS

Capítulo 1

LINUX

¿Qué es Linux?

LINUX es un sistema operativo basado en Unix, su kernel fue desarrollado inicialmente por Linus Torvalds, en 1991. Este sistema es de libre distribución porque está licenciado bajo la GPL[1] v2.

Actualmente recibe el aporte de programadores de todo el mundo. Todo el desarrollo se publica en la *Linux Kernel Mailing List Archive*[2]. Una de las ventajas del núcleo de Linux es su portabilidad a diferentes tipos de computadoras, por lo que existen versiones de LINUX para casi todos los tipos, desde equipos portables, PC domésticas, PC Mac y hasta estaciones de trabajo y supercomputadoras. Este sistema operativo se utiliza junto a un empaquetado de software a la que denominamos *distribución* Linux, por ejemplo Debian.

La enorme flexibilidad de LINUX y su gran estabilidad han hecho de este sistema operativo una opción para tener en cuenta por aquellos usuarios que se dediquen a trabajar a través de redes de datos, naveguen por Internet, o se dediquen a la programación. GNU/Linux, corre en grandes servidores de todo el mundo.

Historia[3]

El proyecto GNU, lo inició en 1983 Richard Matthew Stallman (rms), y tiene como objetivo el desarrollo de un sistema operativo completo similar a Unix y compuesto enteramente de software libre, sin restricciones de acceso o de desarrollos futuros. El término GNU proviene de «GNU No es Unix». Stallman también inventó el concepto de copyleft[4].

La palabra «libre» se refiere a la libertad de acceder a los códigos fuente y no al precio. Se puede o no pagar un precio por obtener software de GNU. De cualquier manera, una vez que se obtiene el software, se tienen cuatro libertades específicas para usarlo: la libertad de ejecutar el programa como se desee; la libertad de copiar el programa y reproducirlo; la libertad de cambiar el programa como se desee mediante el acceso completo al código fuente; la libertad de distribuir una versión mejorada, ayudando así a construir la comunidad del sistema.

El Comienzo

La aparición de LINUX en el mundo de la informática fue una evolución de la cultura de Unix. Este se desarrolló a mediados de los ’70 cuando los miniordenadores y los grandes sistemas dominaban el mundo corporativo. El problema histórico de Unix ha sido su inaccesibilidad a los programadores y desarrolladores que querían trabajar con él fuera de los grandes sistemas de computadoras. Aunque posteriormente aparecieron versiones de Unix para PC, las primeras versiones comerciales costaban más que el PC en el que se debían ejecutar. Esto facilitó el nacimiento de LINUX, de la mano de Linus Torvalds, un estudiante de la universidad finlandesa de Helsinki, quien, en 1991, se abocó a la tarea de reemplazar a Minix, un clon de Unix de pequeñas proporciones y finalidad académica, desarrollado años antes por Andrew Tannenbaum.

En un principio Linus Torvalds escribió todo LINUX en Assembler, pero luego, llegada la hora de escribir algunos drivers, comenzó a utilizar C, con lo que notó una importante aceleración en los

[1]GLP- licencia de derecho de autor más ampliamente usada en el mundo del software libre y código abierto y garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software.

[2]https://lkml.org/http://lkml.iu.edu/hypermail/linux/kernel/

[3]https://www.gnu.org/gnu/gnu-history.es.html

[4] método para licenciar software de tal forma que su uso y modificación permanezcan siempre libres y queden en la comunidad de usuarios y desarrolladores.

tiempos de desarrollo. A fines de Agosto de 1991, Torvalds ya tenía una precaria versión 00.1 de Linux, que era capaz de montar disquetes y contaba con un pequeño sistema de archivos, pero no fue anunciada como oficial ya que necesitaba de Minix para compilarla. En Octubre del '91 fue anunciada oficialmente la versión 0.02, esta versión podía ejecutar las utilidades bash, gcc, gnu-make, gnu-sed y compress. Esta versión no era muy usable.

A medida que avanzaba en su desarrollo, Linus Torvalds fue dejando el código fuente de las sucesivas versiones del kernel (núcleo) y utilidades de Linux a disponibilidad de los usuarios de Internet. Entonces liberó el núcleo Linux bajo los términos de la GPL, completando un sistema GNU (completo y operativo), el sistema operativo GNU/Linux. Este fue sin duda un gran acierto, ya que hizo posible que una gran multitud de desarrolladores de todo el mundo se familiarizaran con el código, lo cual significó un gran aporte de sugerencias, evolucionando luego hacia el desarrollo distribuido de software, que ha permitido a Linux alcanzar un alto nivel de desarrollo y madurez, así también como un amplio grado de aceptación. De esta asociación nace la llamada GNU/Linux.

Aunque el Linux actual se ha desarrollado con el aporte de programadores de todo el mundo, Torvalds aún mantiene el control sobre los cambios que deban realizarse en el kernel.

Esta es la mascota oficial de Linux, que fue elegida por el creador, para representar el sistema operativo que él había creado. Hoy en día todo el mundo asocia a este simpático pingüino con el sistema operativo Linux.

Características

Multitarea real: la palabra multitarea describe la capacidad de ejecutar muchos programas al mismo tiempo sin detener la ejecución de cada aplicación. La multitarea que usa LINUX llamada *preferente* administra los recursos garantizando que todos los procesos en ejecución, incluidos los que corren en background, tengan su tiempo de procesador.

Multiusuario: LINUX es un sistema operativo multitarea y, obviamente, debe ser multiusuario, es decir que permite a distintas personas acceder al sistema compartiendo los recursos que el Administrador del Sistema (superusuario, root) le asigne de acuerdo con su jerarquía, dándole además a cada uno la privacidad y protección necesarias. Esta característica permite que más de una persona pueda trabajar en la misma versión de la misma aplicación de manera simultánea.

Diseño modular del kernel: solo está presente en memoria un kernel mínimo, cuando se requiere algún servicio o se instala hardware nuevo, se carga dinámicamente en memoria un *módulo kernel*. Los módulos de kernel son cargables *on-demand* (bajo demanda) y no es necesario reiniciar el sistema.

Soporta consolas virtuales: lo que permite tener más de una sesión abierta con el mismo u otro nombre de usuario y conmutar entre ellas fácilmente, con las teclas ALT+F1 a la F7. Soporta 6 consolas de texto y una con entorno gráfico. Soporta los estándares POSIX, BSD, IEEE y System V.

Opera con todos los sistemas de archivos estándar, VFAT de Windows, OS2/FS, IS09660 (CD-ROM), NTFS, ext2, ext3, ext4 y otros.

Sistema Operativo de Red: incluye muchas capacidades para redes y comunicaciones.

Soporte completo de hardware (Portable), multimedia, módems, impresoras, placas de video, monitores, teclados, mouse, etc. El reconocer una amplia variedad de configuraciones de hardware es lo que lo hace portable.

Poderoso entorno gráfico con innumerables sistemas de ventanas. Los más populares son FWVM, GNOME, KDE, CDE, Enlightenment, Afterstep, NextLevel, @rM, Xfce, Lxde, etc.

G.N.U/GLP: si bien Linus Torvalds es el titular del derecho de autor de LINUX, todo, incluido el kernel con sus fuentes, está bajo licencia GNU. Esta licencia permite que todo el software desarrollado bajo este concepto sea de libre distribución, de modo que se ofrece software de calidad al público en general sin que los programadores pierdan sus derechos de autor y dejando abierta la posibilidad para que cualquier otro programador pueda ampliar o modificar el programa.

Librerías compartidas: *shared libraries*, gracias a esta característica, no es necesario que las rutinas contenidas en librerías estándar se carguen más de una vez en memoria, ya que cualquier programa que se encuentre en tiempo de ejecución puede acceder a ellas. De esta manera, los binarios (ejecutables) de LINUX son de menor tamaño y permiten ahorrar espacio en disco y memoria.

Administración de memoria: todos los procesos tienen garantizada una zona protegida de memoria para su ejecución, sin que el mal funcionamiento de una aplicación cuelgue todo el equipo. Define por defecto una partición swap o área de intercambio, en el que almacena los procesos bloqueados y suspendidos, con lo que se garantiza RAM para las aplicaciones activas y en uso.

Aplicaciones: gracias a la licencia GNU, el caudal de aplicaciones disponibles para LINUX crece a un ritmo vertiginoso, especialmente en Internet. Podemos decir que existe software para casi todas las necesidades.

Herramientas de Desarrollo: LINUX es un sistema operativo hecho y pensado por programadores para programadores. A partir de esto, múltiples lenguajes de programación están disponibles bajo Linux. Sin duda el principal de ellos es GNU C/C++, pero también es posible desarrollar en Java, Objective-C, Pascal, LISP, BASIC, Perl, Ada, Eiffel, FORTRAN, Forth, Prolog, Oberon, Simula, Modula-2 y Modula-3, Smalltalk, y algunos otros. También existen varios motores de bases de datos que pueden utilizarse bajo Linux; algunos de ellos son motores relacionales (tales como mBase, Thypoon, MiniSQL, Ingres y Postgres), y otros orientados a objetos (tal como LINCKS). La mayoría de ellos son de carácter experimental o académico, por lo que no igualan las prestaciones de los motores de base de datos comerciales, en especial las relacionadas con performance; sin embargo, el hecho de que su costo sea nulo hace que sea una opción a tener en cuenta al desarrollar un servidor de Intranet de pequeñas proporciones.

Seguridad: en lo referente a seguridad, puede mencionarse que el kernel de Linux tiene el soporte necesario para construir *firewalls* basados en filtrado de paquetes; también existe una versión para Linux de SOCKS, software de firewalling muy popular en los ambientes Unix. A partir del kernel 2.6 se ha integrado al kernel un módulo de seguridad que proporciona el mecanismo para soportar políticas de seguridad para el control de acceso, Security-Enhanced Linux (SELinux).

Convivencia: Linux es capaz de convivir en el mismo disco duro con otros sistemas operativos tales como DOS, Windows u OS/2, permitiendo la selección en el arranque del sistema operativo a *bootear*. Además de soportar su sistema de archivos nativo, Linux tiene soporte para acceder en modo de lectura/escritura a sistemas de archivos FAT (DOS) y VFAT (Windows95) y en modo de solo lectura a sistemas de archivos NTFS (Windows NT).

Sitios en Internet

- Linux Home Page: http://www.linux.org
- Grupo de Usuarios Linux de Argentina: http://www.linux-es.org/enlaces_grupos_usuarios
- Código fuente del núcleo Linux para descarga: http://www.kernel.org
- Distribucion RedHat Linux: http://www.redhat.com
- Distribucion Debian https://www.debian.org
- Distribucion Ubuntu https://www.ubuntu.com
- HOW-TOs: los instructivos pueden obtenerse de los siguientes URLs http://www.tldp.org/HOWTO/HOWTO-INDEX/howtos.html
- Linux Documentation Project: serie de libros sobre Linux; pueden obtenerse de http://es.tldp.org/htmls/proy-guia-admon-sistemas.html

Arquitectura

En la arquitectura de Linux podemos identificar un núcleo o kernel y sobre él una capa de shell o interfaz de usuario. El kernel es la parte del sistema operativo más cercana al hardware de la computadora y se considera como el corazón del sistema.

Las funciones más importantes del kernel, aunque no las únicas, son:

- Administrar la memoria para todos los programas y procesos en ejecución.
- Administrar el tiempo de procesador que los programas y procesos en ejecución utilizan.
- Gestionar el acceso y uso de los diferentes periféricos conectados.

El kernel es modular, es decir que está compuesto por módulos. Y éstos manejan la:

- Gestión de procesos
- Gestión de archivos
- Gestión de memoria
- Gestión de Entrada –Salida
- Interfaz de llamadas al sistema

La *interfaz de llamadas al sistema* recibe los pedidos de los programas de usuario o de programas de biblioteca. El *Subsistema de control de procesos* se encarga de las interrupciones, la planificación de procesos, la comunicación entre los procesos y la gestión de memoria. El *Subsistema de Archivos* intercambia los datos entre la memoria y los dispositivos externos. *Control de Hardware*, son rutinas primitivas que interactúan directamente con el hardware.

No todas las funciones de un determinado módulo deben formar parte fija del kernel, sino que pueden ser cargados como módulos en tiempos de ejecución es decir en forma dinámica. Por esto decimos que el kernel de Linux es *modular*. Durante la configuración del kernel es cuando se determina cuáles de ellas se incorporan en forma fija al mismo y cuáles como módulo.

Los módulos del kernel se guardan en */lib/modules/<versión>* (donde versión corresponde a la versión actual del kernel).

lsmod	comando que muestra el estado de los módulos del kernel que están cargados

Siempre que sea posible se debe aprovechar la posibilidad de usar módulos. Las funciones del kernel que no se necesitan durante el arranque de la computadora, se deben tratar como módulos, de este modo se asegura que el kernel no crezca demasiado, y que ni la BIOS ni ningún gestor de arranque, tenga problemas al cargar el kernel. Un ejemplo claro de las funciones que siempre tienen que formar parte del kernel, es el driver del disco duro, el soporte del sistema de archivos ext4, mientras que el soporte de isofs, msdos o sound siempre se deberían compilar como módulos.

uname	comando que muestra información del sistema

Opciones:

- **-s** muestra el nombre del kernel
- **-r** muestra la revisión (release) del kernel
- **-n** muestra el nombre por el que se identifica el sistema en la red
- **-m** muestra el tipo de arquitectura que se está utilizando
- **-v** muestra la versión del kernel
- **-p** muestra información sobre el procesador o unknown
- **-i** muestra la plataforma de hardware o unknown
- **-o** muestra el nombre del sistema operativo
- **-a** muestra toda la información sobre el tipo de sistema que se está utilizando. En el siguiente orden:

Nombre del kernel – hostname del nodo de red – revisión del kernel – Versión del kernel – nombre de la máquina- tipo de procesador –plataforma del hardware

Sintaxis:

```
uname [opción]
```

Ejemplo 1

```
$ uname –a
Linux wheezy 3.2.0-4-686-pae[5]  #1 SMP Debian 3.2.54-2 i686 GNU/Linux
```

Ejemplo 2

También podemos ver esta información en el archivo */proc/version.*

```
$ uname –m
```

Esta opción devolverá x86_64 (en arquitecturas de 64 bits) o i686 (en arquitecturas de 32 bits).

[5] kernel 3.2 de Linux y los módulos para su uso en PCs con uno o más procesadores que soporten PAE (Physical Address Extension).https://packages.debian.org/es/wheezy/linux-image-3.2.0-4-686-pae

Versión del Kernel

Existen dos versiones del kernel de Linux:

Versión de producción: la versión de producción es la versión estable hasta el momento. Esta versión es el resultado final de las versiones de desarrollo o experimentales.

Cuando el equipo de desarrollo del kernel experimental, decide que ha conseguido un kernel estable y con la suficiente calidad, se lanza una nueva versión de producción o estable. Esta versión es la que se debería utilizar para un uso normal del sistema, ya que son las versiones consideradas más estables y libres de fallos en el momento de su lanzamiento.

Versión de desarrollo: esta versión es experimental y es la que utilizan los desarrolladores para programar, comprobar y verificar nuevas características, correcciones, etc. Estos núcleos suelen ser inestables.

Las versiones del kernel se numeran con 4 números, de la siguiente forma: VV.XX.YY.ZZ

- **VV:** Indica la serie principal del kernel. Este número cambia cuando la forma de funcionamiento del kernel ha sufrido un cambio muy importante. Sólo ha sido modificado cuatro veces: en 1994 (versión 1.0), en 1996 (versión 2.0), en 2011 (versión 3.0), en 2015 (versión 4.0) y en 2019 (versión 5.0).
- **XX**: Hasta la versión 2.5 del kernel indicaba si la versión era de desarrollo o de producción. Un número impar, significa que es de **desarrollo**, uno par, que es de **producción**. A partir del kernel 2.6, esto cambió, ahora indica la revisión del mismo.
- **YY**: Indica nuevas revisiones, en las que se han incorporado nuevas características y drivers.
- **ZZ**: indica que se han realizado parches de seguridad y corrección de errores (bugfixes).

Pueden agregarse también algunas letras como “rc1” o “mm2”. El “rc” se refiere a “release candidate” e indica un lanzamiento no oficial. Otras letras usualmente hacen referencia a las iniciales de la persona. Esto indica una bifurcación en el desarrollo del núcleo realizado por esa persona, por ejemplo ck se refiere a **Con Kolivas**, ac a **Alan Cox**, mientras que mm se refiere a **Andrew Morton**.

Distribuciones

Una distribución es un conjunto de utilerías o herramientas y programas que facilitan el trabajo con el sistema. Las distribuciones se pueden diseñar a partir de diferentes versiones del núcleo, también pueden incluir un conjunto de diferentes aplicaciones, utilidades y controladores, y pueden ofrecer distintos procedimientos de instalación y actualización. También las hay para uso doméstico, empresarial y para servidores.

Actualmente son varias las distribuciones de Linux más difundidas, entre ellas podemos mencionar Debian (mantenido por la comunidad), Slackware , Suse, RedHat, Mandriva, Conectiva, Caldera, Ututo (desarrollo argentino), Goobuntu (basada en Ubuntu que Google usa dentro de sus oficinas como sistema operativo de sus equipos informáticos). El único elemento común entre ellas es el kernel.

Además del núcleo Linux, las distribuciones incluyen habitualmente como segunda capa las bibliotecas C de GNU y herramientas del proyecto GNU y el sistema de ventanas X Window System.

Dependiendo del tipo de usuarios a los que la distribución esté dirigida se incluye también otro tipo de software como procesadores de texto, hoja de cálculo, reproductores multimedia, herramientas administrativas. Las distribuciones son liberadas por sus desarrolladores en forma de código fuente, permitiendo a sus usuarios modificar o compilar el código fuente original si lo desean.

Las distribuciones están divididas en «paquetes». Cada paquete contiene una aplicación específica o un servicio. El paquete es generalmente distribuido en su versión compilada y la instalación y desinstalación de los paquetes es controlada por un sistema de gestión de paquetes. Cada paquete contiene meta-información tal como fecha de creación, descripción del paquete y sus dependencias. El gestor de paquetes permite la búsqueda de paquetes, actualizar las librerías y aplicaciones instaladas, y obtener todas las dependencias.

Algunos de los sistemas de paquetes más usados son:

- rpm creado por Red Hat y usado por un gran número de distribuciones de Linux, es el formato de paquetes del Linux Standard Base.
- debian, paquetes Debian, originalmente introducidos por Debian, pero también utilizados por otros como Knoppix y Ubuntu.
- tar.gz .tgz, usado por Slackware, empaqueta el software usando tar y gzip. Pero, además, hay algunas herramientas de más alto nivel para tratar con este formato: slapt-get, slackpkg y swaret.

Para obtener información sobre la distribución que estamos utilizando, podemos utilizar el siguiente comando:

lsb_release	muestra información de la distribución GNU/Linux del equipo

Sintaxis:

```
lsb_release [opcion]
```

Opciones:

-a, **–all**	Muestra toda la información
-i, **-id**	Muestra el nombre de la distribución GNU/Linux
-d, **–description**	Muestra la descripción de la distribución
-r, **-release**	Muestra la versión de la distribución
-c, **-codename**	Muestra el código de la distribución
-s, **-short**	Muestra la información solicitada en formato corto, sin el título inicial.
-h, **-help**	Muestra la ayuda y finaliza
-v, **-version**	Muestra la versión de la especificación LSB de la que es compatible y finaliza

Ejemplo 1

```
$ lsb_release -a
No LSB[6] modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 15.04
Release:    15.04
Codename:vivid
```

[6]LSB linux standard base: comprende un conjunto de estándares que se deben respetar para que una aplicación pueda ejecutarse en cualquier distribución Linux.

Ejemplo 2

```
$ lsb_release-id
Distributor ID:  Debian
Description:    Debian GNU/Linux 8.3 (jessie)

$ lsb_release --codename
Codename:   jessie
```

También podemos consultar el archivo */etc/os-release*. Tipeamos el siguiente comando:

```
$ cat/etc/os-release

NAME="Ubuntu"
VERSION="15.04   (Vivid Vervet)"
ID= Ubuntu
ID_LIKE= debían
PRETTY_NAME= "Ubuntu 15.04"
VERSION_ID= "15.04"
HOME_URL= "http//www.ubuntu.com/"
SUPPORT_URL="http//help.ubuntu.com/"
BUG_REPORT_URL="http//bugs.launchpad.net/Ubuntu/"
```

Tipos de Shell

LINUX dispone de varios programas que se encargan de interpretar los comandos que introduce el usuario y realiza las acciones oportunas en respuesta, estos programas se denominan *shell*. El shell es capaz de interpretar una gran gama de comandos y sentencias. Permite construir programas y comandos, llamados *shellscripts*, que nos dan la posibilidad de automatizar diversas tareas.

Existen varios tipos de *shells* orientados a caracteres (modo texto). A estos shells los encontramos como archivos ejecutables en el directorio */bin***.** Los shells más utilizados son

- *ash*: *A shell*
- *csh*: *C shell*
- *tcsh*: una extención al *C shell*
- *ksh*: *Korn Shell*
- *bsh*: *Bourne Shell*
- *bash*: *Bourne Again Shell*

El caracter indicativo de Bourne y Korn shell es el signo **$.** El signo **%** se refiere a C shell, estando reservado el símbolo **#** para la cuenta de administrador. El más utilizado por las distintas distribuciones Linux, es *bash*. Toma las facilidades de *csh* y *ksh*. Ofrece entre otras posibilidades las siguientes:

- **Autocompletado durante la escritura**
  Al teclear uno o varios caracteres se puede pulsar TAB con el objetivo de que en caso de que pueda completarse de forma unívoca un comando, nombre de archivo o una variable (en dependencia del contexto), se complete de forma automática (se escriba el resto de la palabra).

- **Historial de comandos**
  Esta es una facilidad de muchos otros *shells* que permite el movimiento (con las teclas de cursor arriba y abajo) a través de los últimos N comandos ejecutados, en la sesión actual o en las anteriores. N por defecto es 1000, pero puede modificarse.

- **Estructuras de control**
  if, for, while, select y case

- **Definición de funciones y alias para comandos**
  Las funciones permiten definir subrutinas y los alias, asociar nombres a comandos con ciertas opciones y argumentos de forma más nemotécnica o abreviada.

El usuario puede cambiar el shell asignado por defecto simplemente tipeando el nombre de éste.

Ejemplo

```
$ csh     <Enter>
%         < Aparece el prompt % de C shell.>
```

Formato de la línea de comandos

Para Linux, ***un comando es cualquier archivo ejecutable***. Por lo que cada usuario puede construir archivos con una estructura especial que le permita ser ejecutados, y añadirlo a su lista de comandos. Podrá ejecutar comandos desde una terminal de texto.

Linux utiliza un formato sencillo para la línea de comandos, en donde la mayoría de ellos está formado por sólo dos letras.

Sintaxis:

```
comando [-opciones ] argumento1 argumento2
```

- El nombre del comando es siempre necesario.
- Los comandos siempre van escritos en letra minúscula y la mayoría de las opciones también.
- Las opciones son caracteres o adverbios que siempre van precedidos de un guión.
- Los argumentos pueden o no ser necesarios, sin embargo los nombres de archivos que especifique pueden ser rutas de acceso relativas o absolutas. Por ejemplo: *ls **../../dev***
- En los argumentos podemos utilizar metacaracteres, como son ***** y **?**.
- Los espacios son importantes ya que son separadores de campo por defecto.

Ejemplo

```
$ ls -a -color=yes /usr/conf*
```

El comando anterior lista todos los archivos, ocultos y no ocultos, cuyo nombre comience con las letras *conf*, diferenciando con colores los distintos tipos de archivos.

No olvide que los nombres de los archivos de Linux incluidos los comandos distinguen las mayúsculas de las minúsculas. El sistema ordena alfabéticamente las mayúsculas antes que las minúsculas.

Interfaces de usuario

Una vez instalado e inicializado el sistema operativo Linux, se dispone de dos vías fundamentales de interacción con el usuario: una Interfaz de Línea de Comandos con muchísima potencia conocida como Consola de texto y una Interfaz Gráfica de Usuario, GUI, que es un programa en ejecución o proceso lanzado desde la línea de comandos tras arrancar el equipo.

GUI - Interfaz Gráfica de Usuario

La interfaz gráfica se implementa a través del sistema de ventanas estándar *X Window* que es un entorno operativo gráfico que permite dar soporte a aplicaciones en red y muestra la información gráfica de forma totalmente independiente del sistema operativo. La arquitectura de un sistema *X* es un protocolo cliente-servidor.

Cuando esta interfaz pertenece al propio sistema operativo debemos hablar de **"entorno de escritorio" o DE (Desktop Environment)** que es un conjunto de software que ofrece una interfaz amigable entre el usuario y el sistema operativo. Un entorno de escritorio ofrece facilidades de acceso y configuración, como barras de herramientas, íconos, carpetas, fondos de pantalla y widgets de escritorio e integración entre aplicaciones con habilidades como arrastrar y soltar por ejemplo. Entornos de escritorio hay muchos, por ejemplo **GNOME, KDE, CDE, Xfce** o **LXDE** que son de código abierto (o software libre) y comúnmente usados en distribuciones **Linux**.

Los entornos de escritorios por lo general no permiten el acceso a todas las características que se encuentran en un sistema operativo. En su lugar, la tradicional interfaz de línea de comandos llamada también CLI, muchas veces se utiliza cuando se requiere el control total sobre el sistema operativo.

Un **Gestor de Ventanas** tiene asociadas las acciones de abrir, cerrar, minimizar, maximizar, mover, escalar y mantener un listado de las ventanas abiertas. Es también muy común que el gestor de ventanas integre elementos como: el decorador de ventanas, un panel, un visor de escritorios virtuales, íconos y un tapiz. Entre los Gestores de Ventanas más conocidos están: **AfterStep, Kwin** (para KDE), **Metacity** (paraGnome), **FVWM, AmiWM** (Amiga Windows Manager), **IceWM, Openbox** (de LXDE), y otros tantos. El sistema gráfico X Window, de GNU/Linux, permite al usuario elegir entre varios gestores.

CLI- Interfaz de Línea de Comandos

GNU/Linux se caracteriza porque es un sistema operativo multiusuario, que permite que varios usuarios puedan estar trabajando de manera independiente en la misma máquina a través de las diferentes consolas.

Para esto Linux ofrece el mecanismo de Consolas de texto, también llamadas CLI. Este consiste en que a partir de una entrada (el teclado) y con una salida (el monitor) se simulen varias terminales, donde el mismo, o distintos usuarios puedan conectarse indistintamente. De esta forma, es posible tener más de una sesión abierta en la misma máquina y trabajar en ellas indistintamente.

En concreto, cualquier sistema GNU/Linux dispone de varias Interfaces de línea de comandos o Consolas de texto, disponibles para iniciar sesiones de usuario a las que se accede mediante las teclas Crtl+ALT+F1, Crtl+ALT+F2, Crtl+ALT+F3, Crtl+ALT+F4, Crtl+ALT+F5, Crtl+ALT+F6, en esas

consolas podremos utilizar una o diferentes Shell y en cada una de ellas se debe loguear un usuario y se encuentra asociada a un dispositivo tty.

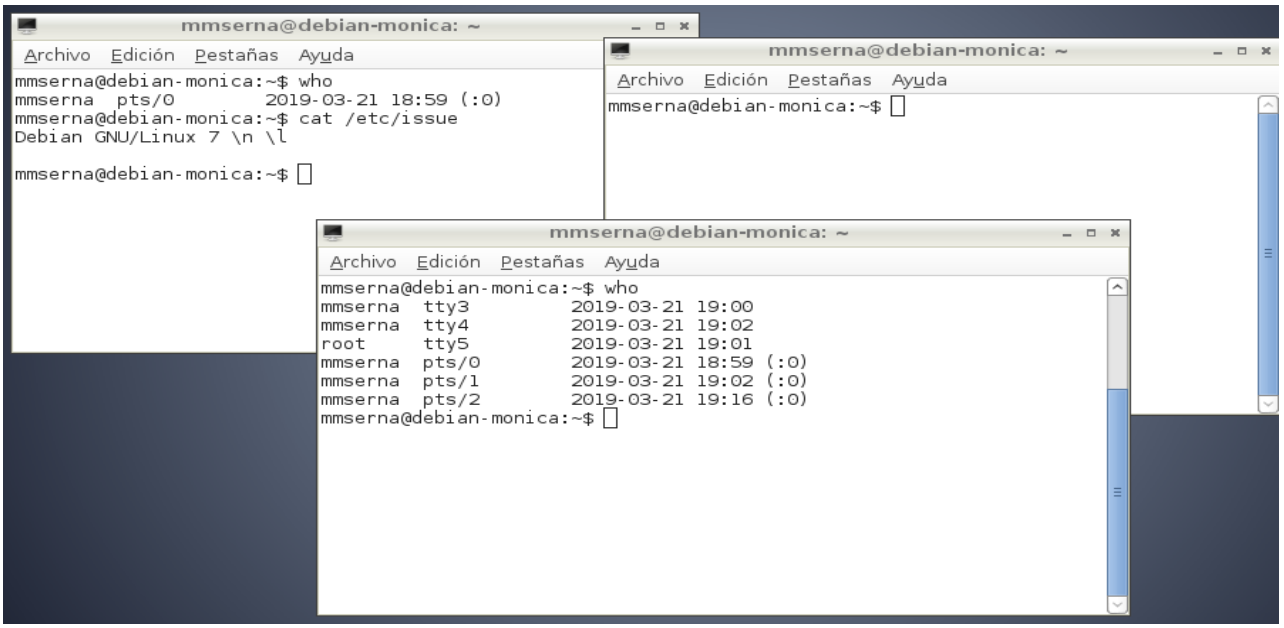
Debian GNU/Linux 7 wheezy tty2

Wheezy login:

Estando en modo texto podemos presionar Crtl+ALT+F7 accedemos a una interfaz gráfica por defecto, en donde podemos tener instalado un Entorno de Escritorio (DE) GNOME, KDE, Xfce, LXDE u otro. Desde el DE podemos ir a Inicio → Herramientas del sistema → Terminal (*emulador de terminal*) o podemos cliquear la opción XTerm (*standard terminal emulator for de X Window system*). Es posible abrir terminales con ALT+F2 y ejecutar por ejemplo *xterm*. Son emuladores de terminal: Gnome Terminal, Konsole, LXterm. Estas terminales de texto al ejecutarse se asocian a los archivos *pts* que están en el directorio */dev.*

Ejemplo

```
$ who -a
```



El usuario mmserna tiene abierto 3 sesiones de trabajo en Consolas virtuales (modo texto) tty3, tty4, tty5 y una sesión de trabajo con entorno grafico (tty7) en la cual tiene abierto 3 terminales de texto asociadas a los archivos pts/0, pts/1 y pts/2.

Actividad 1

1. Enumere las características principales del SO Linux.

2. El núcleo del sistema Linux, se denomina:

a. Shell **b**. BIOS **c**. Kernel **d**. File System **e.** Ninguna

3. Marque cuáles de los siguientes son shells de Linux:

a. msh **b.** ash **c.** bsh **d.** bash **e.** csh **f.** xsh **g.** ksh **h.** fsh

4. Cuáles son las interfaces posibles de Linux?

Actividad 2

1. Los módulos del Kernel se guardan en el directorio */lib/modules/<versión>* **V** ó **F**

2. Explique a qué hace referencia <versión>, del punto anterior.

3. Todas las funciones de un determinado módulo deben formar parte fija del kernel. **V** ó **F**

4. Las siguientes son distribuciones de Linux:
a. RedHat
b. Conectiva
c. Karatte
d. Caldera
e. AutoLink
f. Suse

5. Muestre qué distribución Linux tiene instalada.
`lsb_release -a` ó`cat /etc/os-release`

6. Muestre información sobre el S.O. instalado.
`uname -a`

7. ¿Cuál es la versión del kernel que utiliza su distribución?
`uname -v`

Capítulo 2

Entrada al Sistema

El concepto de arranque

El bootstrapping o proceso de arranque hace referencia al proceso de inicio de la computadora. Cuando encendemos nuestra computadora toma el control de la misma la BIOS (Basic Input/Output System), rutina que se encuentra almacenada en una memoria con el mismo nombre. Luego se ejecuta POST (Power On Self Test), rutina que verifica y dimensiona la memoria RAM y ROM, verifica también la comunicación con los dispositivos conectados al sistema y por último carga el *registro maestro de arranque* MBR (primer sector del disco duro) que contiene la tabla de particiones y generalmente el gestor de arranque.

El bootloader o gestor de arranque es un programa que identifica las unidades booteables y carga el Sistema Operativo (kernel) en multi-etapas, en las que varios programas pequeños se llaman unos a otros hasta que el ultimo carga el SO. Son gestores de arranque múltiple en Linux: LILO[7] y GRUB. Por ejemplo, Ubuntu utiliza el gestor de arranque GRUB, y el código que está en el MBR ejecuta el archivo */boot/grub/core.img*.

La imagen[8] del kernel está en el archivo */boot/vmlinuz-version*, donde versión es el número que identifica la versión del kernel. Por ejemplo */boot/vmlinuz-3.2.0-4-686-pae.* En este archivo está el código que permite que el kernel se descomprima así mismo. Durante la inicialización del Kernel:

- Detecta la cpu y su velocidad, lee los valores predeterminados de la BIOS e inicializa las interfaces elementales de la placa base.
- Inicializa el sistema de gestión de memoria virtual (swapper).
- Realiza la carga del Files-system, es decir que monta la partición raíz al directorio */*.
- Después de ejecutar una serie de procesos internos, llama al proceso **init**.

init (abreviatura de initialization) es el primer proceso en ejecución tras la carga del kernel y el que a su vez genera todos los demás procesos. El proceso *init* tiene PID[9] igual a 1, y lo podemos considerar como el padre de todos los procesos. Es el que realiza realmente la inicialización del sistema. Como lo ejecuta directamente el kernel, *init* no puede ser interrumpido ni aún por la señal 9, que se utiliza para interrumpir cualquier otro proceso.

Init lee el archivo de configuración */etc/inittab* desde donde ejecuta diferentes procesos, incluyendo los procesos **rc**. Existen varios rc ubicados en /etc. Estos procesos ejecutan comandos de inicialización como por ejemplo montar sistemas de archivos, inicializar el espacio de intercambio (swap), configurar la red y arrancar los demonios; es decir, lleva al sistema a un estado en el que puede recibir órdenes de los usuarios. Por ejemplo, */etc/rc.d/init.d/network* arranca las interfaces de red.

En cada terminal virtual, *init* ejecuta un proceso */etc/getty*, que dispone la terminal para que el usuario pueda conectarse. Luego *getty* llama al proceso *login*, el cual permite abrir una sesión de trabajo.

Gestor de arranque GRUB (Grand Unified Bootloader)

GNU/GRUB es un gestor de arranque múltiple, desarrollado por el proyecto GNU que se usa comúnmente para iniciar uno, dos o más sistemas operativos instalados en un mismo equipo. GRUB se usa principalmente en sistemas operativos GNU/Linux. En 1999 pasó a ser un paquete del proyecto GNU en su versión más utilizada que es la GRUB Legacy.

El desarrollo de una versión más moderna es GRUB 2, ya que el código fue reescrito en forma completa. GRUB 2 realiza la carga del sistema operativo en dos fases. En la fase 1, la BIOS carga y

[7] LILO "Linux Loader" fue discontinuado a partir de diciembre del 2015. LILO permite seleccionar entre 16 imágenes en el arranque. Para configurar LILO edite el archivo */etc/lilo.conf*

[8] El código fuente del kernel siempre se encuentra en */usr/src/linux*, hacemos uso de este código cuando compilamos algunos programas.

[9] PID identificador de proceso. Es un número que identifica al proceso hasta que muere.

ejecuta el código de la partición de arranque del dispositivo de arranque designado, que se encuentra en el MBR y carga el código que reconoce el sistema de archivos. En la fase 2, le presenta al usuario un menú de inicio desde el cual puede elegir el SO con el que desea bootear.

Niveles de ejecución

Un nivel de ejecución es un estado de init y de la totalidad del sistema, y define qué programas y servicios se ejecutan al iniciar el sistema. Los niveles de ejecución son identificados por números. En general los GNU/Linux tienen definidos 7 runlevels, del 0-6 y S (single mode):

Runlevel	Descripción
0	Halt. Apagar el sistema
1	Modo monousuario (modo rescate)
2 al 5	Modo Multiusuario
6	Reboot. Reinicio del sistema

Los niveles de ejecución S[10] (single) y 1 se utilizan para el mantenimiento del sistema en monousuario. Se inicia la mínima cantidad de servicios para evitar posibles problemas. Por defecto Debian no hace diferencia entre los niveles 2 a 5, salvo que se especifique. Por lo tanto en Debian, del 2 al 5 es modo multiusuario.

El Runlevel estándar, en el cual arranca el sistema, está definido en el archivo */etc/inittab***,** en *initdefault y* dependerá de la distribución que se haya instalado (en el caso de Red Hat 6.2 el nivel es por defecto el 5). En el caso de Debian el runlevel predeterminado es 2 modo multiusuario.

```
:
# The default runlevel
id:2:initdefault:
```

Los runlevels están organizados en directorios en */etc:*

```
/etc/rc0.d      runlevel 0
/etc/rc1.d      runlevel 1
/etc/rc2.d      runlevel 2
/etc/rc3.d      runlevel 3
/etc/rc4.d      runlevel 4
/etc/rc5.d      runlevel 5
/etc/rc6.d      runlevel 6
```

Cada uno de estos directorios tiene un conjunto de enlaces simbólicos a los archivos del directorio */etc/init.d*. Cada uno de estos directorios tiene un conjunto de script que controlan la detención o el inicio de los servicios. Por ejemplo los servicios del runlevel 1 son:

```
/etc/rc1.d:
K01alsa-utils     K01network-manager           K06nfs-common     So1motd
K01atd            K01saned                     K06rpcbind        S02single
K01bluetooth      K01virtualbox-guest-utils    README
K01exim4          K02avahi-daemon              S01bootlogs
K01lightdm        K04rsyslog                   S01killprocs
```

[10]Modo Monousuario: sin interfaz de red ni demonios. Solamente está activo el usuario root, sin contraseña. Este nivel de ejecución permite reparar problemas o hacer pruebas en el sistema.

El nombre de los enlaces respeta la siguiente estructura: **[K | S] nn [string].**

El nombre puede comenzar con la letra K o la S. La *K* indica que el servicio será detenido (kill) al iniciar el runlevel. La letra *S* indica que el servicio se inicia (start). El número *nn* indica la prioridad del servicio dentro del runlevel y luego el nombre del servicio. Si los servicios tienen el mismo número de prioridad se procede en orden alfabético.

runlevel muestra el último runlevel que fue ejecutado y el actual.

El comando lee esta información del archivo */var/run/utmp*. Si no hubo cambio de nivel de ejecución desde que se inició el sistema, muestra la letra N (none).

Ejemplo

```
Runlevel
N  2
2 es el runlevel actual
```

init permite iniciar el sistema en un runlevel determinado. También permite cambiar el nivel de ejecución.

Sintaxis:

```
init  [opción] numero
```

Ejemplo

```
# init  1
```

Al realizar un cambio de Runlevel, se ejecutan los programas de parada del Runlevel actual, por lo que los procesos que se estaban ejecutando finalizan, luego se inicializan los programas de arranque del nuevo Runlevel. En el Runlevel 1 el sistema se inicia en modo monousuario, en donde la máquina no tiene recursos de red, y solo admite un único usuario. Este modo de trabajo es muy útil cuando el supervisor (usuario root) necesita realizar tareas de mantenimiento, sin tener que preocuparse de que pueda haber otros usuarios realizando tareas.

En algunas versiones cuando sucede un fallo general de sistema, éste se reinicia automáticamente en modo monousuario. Pero a partir del 2011 algunas distribuciones habilitan ***systemd*** por defecto, tal es el caso de Fedora, Debian GNU/Linux desde la versión 8 "Jessie, Mandriva 2011, openSUSE 12.1 y superior, Arch Linux desde octubre de 2012,CentOS 7 desde julio de 2014 y Ubuntu a partir de su versión 15.04 (abril de 2015).

Systemd es un software que realiza la administración del sistema y de servicios para Linux. Se desarrolló para reemplazar el sistema de inicio (init) y los niveles de ejecución, por lo que systemd es el padre de todos los procesos. Se diseñó con el objetivo de mejorar el *framework* para expresar dependencias, permitir que se realizara más procesamiento en concurrencia o en paralelo durante el arranque (*booting*) del sistema y reducir el *overhead* del shell. Además del demonio init, systemd, incluye a los *daemons* journald, logind y networkd, junto a otros componentes de bajo nivel. Systemd también integra muchos otros servicios, manejando entradas de usuario, la interfaz de línea de comandos, la conexión en caliente de dispositivos, registro de ejecución programada (remplazando a cron), *hostname* y configuraciones regionales.

Cierre del sistema

Antes de apagar la computadora (servidor) deberá ejecutar el comando ***shutdown,*** que envía un aviso de cierre del sistema a los usuarios que están conectados, y envía una señal a los procesos activos, dándoles tiempo para almacenar sus datos. Cumplido éste tiempo los mata con la orden *kill*.

No es conveniente apagar la computadora sin la ejecución de esta orden, pues debemos dar tiempo al núcleo del sistema operativo de guardar en el disco toda la información que está en memoria, como por ejemplo, archivos abiertos y actualizados. Linux también mueve y actualiza archivos básicos del sistema, que al apagar directamente la máquina podrá dañar éstos archivos y el sistema Linux tendrá problemas luego para ejecutarse.

shutdown apaga o reinicia la computadora.

Sintaxis:

```
shutdown  [ -opciones] [tiempo] [wall mensaje de aviso ]
```

Opciones:

-h apaga la computadora.
-r reinicia la computadora.
+n minutos
hh:mm hora de 0 a 24 y minutos de 0 a 59

El mensaje de aviso de cierre del sistema comienza por “Broadcast Message …” a todos los usuarios conectados. Luego por defecto, shutdown hace una pausa de 60 segundos antes de continuar con el proceso de desconexión. Podemos cambiar este intervalo de tiempo con la opción **–h** más el número de minutos de espera:

Ejemplo 1

```
# shutdown -h +10
El sistema se apagará en 10 minutos
```

Ejemplo 2

```
# shutdown -h 22:05
El sistema se apagará a las 22 hs 5 minutos.
```

Esta orden es ejecutada por el usuario supervisor. En su defecto puede hacerlo un usuario con privilegios ejecutando la orden sudo como sigue:

Ejemplo 3

```
$ sudo shutdown –r +5 en 5min se reiniciará el sistema
el sistema le pedirá su clave de usuario[11]. Enviara un mensaje a
todos los usuarios conectados. Luego reiniciará la computadora.
```

Ejemplo 4

```
 # shutdown -h now
 # shutdown -r now
apaga o reinicia ahora el Sistema respectivamente
```

[11]Los usuarios que pueden ejecutar el comando sudo pertenecen al grupo sudo.

Inicio de sesión

Para comenzar a trabajar con el sistema operativo es necesario abrir una sesión de trabajo, para ello deberá introducir el nombre de usuario y contraseña, las cuales lee y verifica el programa*login.* De esta manera el sistema proporciona seguridad, ya que lleva la cuenta de quién es el dueño de los archivos y quién tiene acceso a ellos. Dependiendo de la configuración, la pantalla que aparecerá puede ser en modo gráfico o en modo texto.

login	Realiza la autenticación del usuario y establece una nueva sesión con el sistema.

Sintaxis:

```
login [opción] [username]
```

Opciones:

- **-h** Nombre del host remoto para este inicio
- **-p** preservar el entorno
- **-r** realizar protocolo de inicio de sesión automático para rlogin

Las opciones –r y –h las utiliza root.

El comando *login* es el que pide el nombre de usuario (username) con el cual el sistema operativo identifica una cuenta de usuario. Luego pide la palabra clave, la cual no se visualiza en pantalla. Login toma el *username* y verifica su existencia en el archivo */etc/password*. La clave se verifica contra el archivo /etc/shadow. Luego muestra el contenido del archivo /etc/motd (incluye el mensaje del día) y comprueba si el usuario tiene correo electrónico. Podemos personalizar la entrada al sistema al mostrar un mensaje antes que se ejecute login. El mensaje a mostrar se almacena en el archivo /etc/issue. Generalmente se muestra el nombre del sistema o la distribución, así como su versión. El contenido de este archivo puede ser:

```
Debian GNU/Linux  7 \n  \l
```

Todas las secuencias de escape son precedidas por la barra invertida, en el ejemplo: \n imprime el nombre de la máquina (hostname), mientras que \l imprime el nombre de la tty actual. Por lo que se mostrará en pantalla lo siguiente:

```
Debian GNU/Linux   7 wheezy   tty1

Wheezy login:
```

También podemos utilizar las siguientes secuencias:

- **\d** Inserta la fecha actual.
- **\m** Inserta el identificador de la arquitectura de la máquina, i486 por ejemplo.
- **\o** Inserta el nombre del dominio de la máquina.
- **\r** Inserta la versión del núcleo, por ejemplo 2.4.16.
- **\s** Inserta el nombre del sistema, el nombre del sistema operativo.
- **\t** Inserta la hora actual.

Cuando el usuario introduce la contraseña no habrá eco desde el sistema, por lo que hay que ser cuidadoso a la hora de teclearla.

```
Login: username <Enter>
Password:
```

Una vez que el sistema acepta al nuevo usuario, aparecerá una serie de mensajes en pantalla dependiendo de la configuración del sistema, por último se mostrará el indicativo del shell, el signo **$** o **#** para el usuario root. El usuario comienza su sesión en un directorio de trabajo que le es propio, donde él es el dueño, generalmente */home/nombre-de-usuario* al cual llamamos *directorio de conexión*.

Nombre de archivo	**Información que almacena**
/var/run/utmp	Lista de sesiones abiertas actualmente
/var/log/wtmp	Lista de inicio de sesión anteriores
/etc/passwd	Información de cuenta de usuarios
/etc/shadow	Claves de las cuentas de usuarios
/etc/motd	Mensajes del sistema del día
/etc/nologin	Usuarios no-root que no pueden ingresar
/etc/tty	Lista de tipo de terminales

Tabla 1. Algunos de los Archivos que utiliza el comando login

Login almacena todos los accesos fallidos en el archivo */var/log/btmp,* incluso los accesos de root. Cada vez que inicia una sesión actualiza el archivo /var/run/utmp el cual se limpia cada vez que se inicia el sistema. También registra el acceso en el archivo */var/log/wtmp*, pero este archivo crecerá sin límites, por lo que es necesario limpiarlo regularmente.

who muestra información sobre las sesiones actualmente abiertas.

Sintaxis:

```
who [opción] …[FILE Arg1 Arg2]
```

Opciones:

- **-b** tiempo transcurrido desde el inicio del sistema (boot).
- **-q** conteo de usuarios logueados al sistema.
- **-r** runlevel actual del sistema.
- **-u** usuarios logueados.
- **-a** listado detallado de sesiones con información extendida, tiempo de sistema y runlevel actual.
- **-H** muestra la cabecera de las columnas

Si no se especifica FILE lee desde el archivo */var/run/utmp*

Ejemplo

```
$who -r

'run-level' 5 2017-03-26  16:40
```

last muestra un listado de las últimas sesiones abiertas.

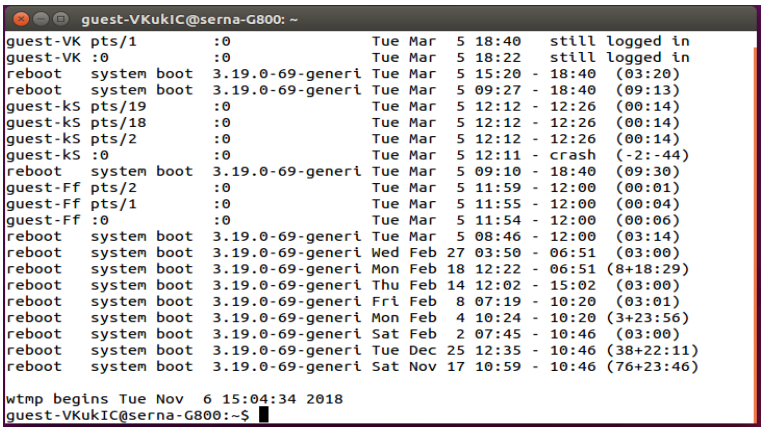
Sintaxis:

```
last [opciones]
```

Este comando muestra el contenido, por defecto, del archivo */var/log/wtmp, en donde se registra el inicio y cierre de sesión de todos los usuarios.*

Ejemplo:

```
$last
```



lastb muestra un listado de los inicios de sesión fallidos.

Sintaxis:

```
lastb [opciones]
```

El comando *lastb* muestra el contenido del archivo */var/log/btmp.*

Ejemplo

```
#lastb

malumno      tty1  Wed  Mar  29  11:54 - 11:54
UNKNOW       tty1  Tue  Mar  21  09:58 - 09:58
```

Cierre de sesión

Cerrar la sesión es muy importante porque, varios usuarios pueden estar compartiendo los recursos del sistema. El sistema lleva la pista del usuario que termina su sesión, entonces cierra sus archivos y finaliza los procesos activos. Si estos procesos quedan funcionando, provocan una carga innecesaria en el sistema y perjudican a otros usuarios. Luego escribe una entrada en el archivo wtmp y borra la sesión del archivo utmp.

logout cierra la sesión de usuario.

Sintaxis:

```
logout
```

Una vez que cierra la sesión llama a login. También puede utilizar el comando *exit* o teclear *ctrl.+d.*

Actividad 1

1. Abra una sesión de trabajo con un nombre de usuario que le asigne su administrador. En este ejemplo utilizaremos el username *usuarioxx*

```
login
```

A continuación aparecerá el prompt del sistema que dependerá del shell que utilice el usuario.

2. Tipee el comando *who* que le permitirá ver que usuarios han iniciado una sesión con el sistema.

La salida de este comando nos muestra, en la segunda columna, la terminal virtual (archivo especial de dispositivo) en el cual está trabajando el usuarioxx

3. Abra otra sesión de trabajo con el username *usuarioyy*

```
Ctrl.+Alt+F2
login: usuarioyy
password:
```

Ejecute el comando who nuevamente.

¿Cuántos usuarios informa el sistema que están conectados? _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

¿Puede distinguir en qué terminales está trabajando cada usuario? _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

4. Comente qué es una terminal virtual.

Actividad 2

1. Regrese al entorno gráfico con Ctrl.+Alt+F7 y abra dos consolas virtuales y ejecute el siguiente comando en la consola activa (la segunda abierta):

```
ls  -R  /home
```

2. Ejecute en la consola numero 1 el siguiente comando

```
ls  -R  /etc
```

En cada una de las consolas virtuales está ejecutándose una tarea distinta, observe que ambas están siendo atendidas por el microprocesador en un pseudoparalelismo, por lo que estamos frente a un ejemplo claro de multitarea. Puede alternar de una consola a otra con el simple movimiento del mouse

3. Ejecute el comando *who* y podrá observar que *usuarioyy* está activo en varias consolas. Identifique en cuántas está activo y en cuáles.

4. Cambie al shell de C y ejecute algunos comandos para familiarizarse con el entorno.

```
csh
```

Actividad 3

1. ¿Qué función cumple el proceso *getty*?
- **a)** Inicializa la terminal
- **b)** Inicializa los drivers
- **c)** Determina el runlevel

2. El Runlevel por defecto es:
- **a)** Monousuario
- **b)** Multiusuario con red y Xwindow
- **c)** Multiusuario sin red

3. El proceso *shutdown* efectúa:
- **a)** Cierre de sesión
- **b)** Termina los procesos y cierra archivos
- **c)** Cierra archivos solamente

4. ¿Cuál es la función del comando *login*?

5. Explique la diferencia entre los comandos *shutdown* y*logout.*
6. Explique cuál es la estructura de la línea de comandos.
7. Al realizar un cambio de Runlevel no es necesario dar de baja a los procesos que se están ejecutando. **V** ó **F**

8. Al iniciarse una sesión de trabajo, qué datos requiere el comando login, y qué archivos utiliza para validar los mismos?

Actividad 4

1- Como usuario administrador de una pequeña red, debe apagar el sistema para realizar tareas de mantenimiento en el hardware del servidor. Por lo tanto tendrá que:
   a- Loguearse como root y verificar si hay usuarios logueados.
   b- Apagar el sistema pero incluyendo un mensaje para los usuarios en forma de preaviso, con el objetivo de que guarden sus archivos.

```
a)  su – -
    who
b)  shutdown  +3  por tareas de mantenimiento el sistema se apagará en 3
minutos.
```

2- Como usuario administrador de una pequeña red, debe realizar tareas de mantenimiento por lo que necesita llevar al sistema al modo de trabajo monousuario. Por lo tanto tendrá que:

a- Verificar si hay usuarios logueados.
b- Enviar un mensaje general a los usuarios informando de las tareas de mantenimiento con el objetivo de que guarden sus archivos.
c- Reiniciar en modo monousuario.

```
      su - -
      who
   a- wall el sistema se reiniciará en monousuario en unos minutos, guarde
      sus datos.
   # el comando wall permite enviar un mensaje a todos los usuarios
   conectados.
 b- init 1
```

Capítulo 3

Conociendo el Sistema de Archivos (File System)

El Sistema de Archivos

El sistema de archivos es la parte del sistema operativo que se encarga de administrar los archivos y directorios de un dispositivo. Entre sus funciones, podemos mencionar la asignación de espacio a los archivos, la administración del espacio libre y el acceso a los datos almacenados. Cada sistema operativo maneja su propio sistema de archivos.

Los discos rígidos son dispositivos de almacenamiento que permiten el acceso a los datos a través de bloques o sectores. El software del sistema de archivos es quien se encarga de organizar estos sectores en archivos y directorios y mantiene un registro de qué sectores están asignados a un archivo en particular y qué sectores están libres. En la práctica, un sistema de archivos también se puede utilizar para acceder a datos generados dinámicamente, como los que se reciben a través de una conexión de red, sin la intervención de un dispositivo de almacenamiento.

Sistemas de archivos de Linux

En Linux (como en cualquier Sistema Operativo) se puede diferenciar el sistema de archivos que está almacenado en las particiones del disco y otros dispositivos, y el sistema de archivos tal y como lo presenta al usuario un equipo ejecutando Linux. En Linux se monta un sistema de archivos de disco en el sistema de archivos del equipo en funcionamiento.

Un disco duro no es más que un gran espacio sobre el que se pueden escribir unos y ceros. Un sistema de archivos impone cierta estructura, y hace que parezca que hay archivos dentro de directorios, y directorios dentro de directorios. Cada archivo se representa por un bloque especial llamado i-nodo (nodo índice), que contiene información acerca de sus atributos, como por ejemplo, quién es el dueño del archivo, fechas de creación, modificación, permisos, etc., como así también dónde encontrar su contenido. Los directorios también se representan por medio de i-nodos, pero éstos contienen información sobre cuáles son los i-nodos de los archivos que están en dicho directorio.

Por ejemplo, si el sistema quiere leer */home/cecilia/imagen1.jpg* primero busca el i-nodo del directorio raíz *"/"*, luego busca el i-nodo del directorio *home* en el contenido de *"/",* luego busca el i-nodo del directorio *cecilia* en el contenido de */home*, luego el i-nodo de *imagen1.jpg* que le dirá qué bloques del disco leer.

La facultad que tienen los directorios de poder contener información sobre otros directorios (denominados subdirectorios) permite una organización o estructura jerárquica con la forma de un árbol invertido. El directorio inicial de esa jerarquía se denomina directorio raíz y se simboliza con una barra de división (*/*).

Cada sistema de archivos pertenece a una clase o tipo que define la característica administrativa. Así como algunas versiones de Windows usan FAT o NTFS, **Linux** en particular adopta el sistema de archivo **Ext4**.

Sistema de Archivos Ext4 (cuarto sistema de archivos extendido - fourth extended file system)

"El ext4 es un sistema de archivos transaccional planteado como una mejora de ext3. Se publicó en el kernel Linux 2.6.28 en diciembre del año 2008. Entre las principales mejoras, podemos encontrar:

- Sistema de archivos de gran tamaño. El sistema de archivos ext4 es capaz de trabajar con volúmenes de gran tamaño, hasta 1 EiB (exibyte = 2^{60} bytes) y archivos de hasta 16 TiB (tebibyte = 2^{40} bytes).
- Soporte de volúmenes de hasta 1024 PiB (pebibyte = 2^{50} bytes)
- Soporte añadido de extensión (extent). Las extensiones se utilizan para reemplazar al tradicional esquema de bloques usado por los sistemas de archivos ext2/3. Una extensión es un conjunto de bloques físicos contiguos, que mejora el rendimiento, cuando se trabaja con archivos de gran tamaño, reduciendo la fragmentación.

- Compatibilidad hacia adelante y hacia atrás. Cualquier sistema ext3 existente se puede montar como **ext4** sin necesidad de modificar el formato del disco. También se puede actualizar un sistema de archivos ext3 para conseguir las ventajas del ext4 ejecutando sólo algunos comandos y sin necesidad de reformatear o reinstalar el sistema operativo. Esto se puede implementar en un sistema en producción en forma segura y sin arriesgar los datos.
- Asignación persistente de espacio en el disco. El sistema de archivos ext4 permite reservar espacio en disco para un archivo. Hasta el momento, el método consistía en rellenar con ceros el archivo, en el momento en que se creaba. Esta técnica ya no es necesaria con ext4, porque existe una llamada al sistema denominada "preallocate()". El espacio reservado para estos archivos queda garantizado y con probabilidad de ser contiguo. Esta función es útil en aplicaciones de streaming y bases de datos.
- Límite de 64.000 subdirectorios
- Desfragmentación en línea
- Menor uso del CPU
- Mejoras en la velocidad de lectura y escritura"

Sistema de archivos Btrfs

"Btrfs es un sistema de archivos desarrollado por Oracle Corporation, lanzado en Junio del año 2007 para GNU/Linux. Btrfs es un sistema de archivos de copia en escritura (copy-on-write), diseñado con funciones avanzadas, destacando la tolerancia a fallos, la reparación y una administración sencilla.

El sistema de archivos Btrfs brinda las siguientes características:

- La funcionalidad copia en escritura permite crear instantáneas tanto de lectura y escritura, para poder recuperar un sistema de archivos a un estado anterior, incluso después de que se haya actualizado desde un sistema de archivos ext3 o ext4.
- La funcionalidad de suma de comprobación garantiza la integridad de los datos.
- La compresión transparente ahora espacio en disco.
- La desfragmentación transparente mejora el rendimiento.
- La gestión de volumen lógico integrado permite implementar configuraciones RAID 0, RAID 1 o RAID 10 y añadir y eliminar capacidad de almacenamiento de forma dinámica."

Sistema de archivos ZFS

ZFS es un sistema de archivos desarrollado por Sun Microsystems para su sistema operativo Solaris. Su lanzamiento inicial fue en noviembre del año 2005. Una de sus características principales es que ofrece integridad de datos, garantizando que la información grabada en disco sea siempre correcta, mediante la utilización de un modelo transaccional. Las características principales del sistema de archivos ZFS son:

- Capacidad: 128 bits (2^7 veces la capacidad de un sistema de archivos de 64 bits).
- Auto reparación. ZFS permite configurar agrupaciones de discos denominadas RAID-Z. Cuando se utiliza esta tecnología RAID, se puede detectar y recuperar frente a errores, mediante la utilización de funciones hash asociadas a los datos. ZFS brinda configuración de RAID-Z con tolerancia a fallos de paridad simple, doble o triple. RAID-Z de paridad simple (RAIDZ o RAIDZ1) es similar a RAID-5 y de paridad doble (RAIDZ2) es similar a RAID-6.
- Modelo transaccional. ZFS utiliza un modelo transaccional copia en escritura (copy-on-write). Todos los punteros a bloques de un sistema de archivos contienen un checksum de 256 bits sobre el bloque apuntado, que se comprueba al leer el bloque. Los bloques que contienen datos activos no se sobrescriben nunca; en cambio, se reserva un nuevo bloque, y los datos modificados se escriben en él. De esta forma, cualquier bloque de metadatos que lo referencie se reubica y escribe.

- Instantáneas (snapshots). Una instantánea es una imagen de un sistema de archivo en un momento determinado. Esta imagen es de solo lectura y permite disponer de los datos en un momento determinado.
- Bandas de tamaño variable. ZFS permite expandir las bandas de un zpool con o sin redundancia, a medida que se agreguen dispositivos a un pool. Esto permite repartir la carga de escritura de forma equitativa entre todos los dispositivos, optimizando los procesos de lectura (leyendo desde distintas ubicaciones de cada bloque en cada dispositivo).
- Para las aplicaciones, ZFS es un sistema de archivos estándar POSIX; no es necesario ningún cambio en las aplicaciones para guardar datos en ZFS.

Organización de los directorios

Debian GNU/Linux se basa en el estándar FHS (Estándar de Jerarquía de Sistema de Archivos - Filesystem Hierarchy Standard). El objetivo de este estándar de jerarquía del sistema de archivos es normalizar o estandarizar los directorios principales y sus contenidos en las diferentes distribuciones de Linux. El directorio raíz se representa por una barra vertical */.* Todos los sistemas Debian incluyen los siguientes directorios partiendo del directorio raíz.

Directorio	**Contenido**
bin	Comandos principales del sistema.
boot	Archivos estáticos utilizados por el cargador de arranque
dev	Archivos de dispositivos
etc	Archivos de configuración específicos del equipo
home	Directorios de los usuarios
lib	Bibliotecas compartidas esenciales y módulos del núcleo
media	Puntos de montaje para medios extraíbles
mnt	Punto de montaje temporal para un sistema de archivos
opt	Paquetes de programas y aplicaciones opcionales instalados manualmente
proc	Directorio virtual que contiene información del sistema
root	Directorio del usuario administrador del equipo
sbin	Comandos principales del sistema
srv	Datos de los servicios ofrecidos por el sistema
sys	Directorio virtual que contiene la información del sistema
tmp	Archivos temporales
usr	Jerarquía secundaria
var	Datos variables

El uso del disco varía mucho dependiendo de las características y función de cada empresa u organización en particular. A continuación, se sugieren aspectos a tener en cuenta con respecto al contenido de los principales directorios.

La partición raíz */* siempre debe contener físicamente las particiones */etc, /bin*, */sbin*, */lib y /dev,* para que el sistema pueda arrancar correctamente.

/bin Contiene todos los comandos que utilizarán los usuarios, por ejemplo, ls, more, cp, pwd, who, mkdir, rm, ln, etc.

/boot	Contiene todos los ejecutables y archivos relacionados con el arranque del sistema, por ejemplo, el kernel, initrd, gestor de arranque LILO o GRUB. Normalmente este directorio se encuentra en una partición aparte
/dev	Contiene todos los dispositivos de almacenamiento, representados como archivos, que estén conectados al sistema
/etc	Contiene archivos de configuración del sistema específicos del host de todo el sistema. Por ejemplo /etc/shadow, /etc/services, /etc/crontab
/home	*/home* es el directorio de los usuarios del sistema. En él se almacenan todos los archivos del usuario, como documentos, presentaciones, videos, música, etc. Cada usuario tiene su propio directorio. Por ejemplo */home/luz*, */home/pedro*, donde a su vez, cada uno de ellos puede crear su propia estructura de directorios para organizar su información
/lib	Contiene las bibliotecas esenciales para que se puedan ejecutar correctamente todos los binarios ubicados en los directorios */bin* y */sbin*, como así también los módulos del propio kernel. En */lib64* se ubican las bibliotecas para las aplicaciones de 64 bits
/media	Contiene los puntos de montaje de los medios extraíbles de almacenamiento, tales como lectores de CD-ROM, pendrives (memoria USB), e incluso sirve para montar otras particiones del mismo disco duro, como por ejemplo, alguna partición que sea utilizada por otro sistema operativo. Si hay varios usuarios trabajando en un sistema, los puntos de montaje de los volúmenes que montan cada uno de ellos se mostrarán en directorios separados, por ejemplo */media/luz*, */media/pedro*
/mnt	Sistema de archivos montado temporalmente. Es un directorio semejante a */media*, pero lo utilizan los usuarios normalmente. Sirve para montar discos duros y particiones de forma temporal en el sistema; no necesita contraseña, a diferencia del directorio */media*
/opt	Contiene paquetes de programas opcionales de aplicaciones estáticas, es decir, que se pueden compartir entre los usuarios. Estas aplicaciones no guardan sus configuraciones en este directorio. De esta forma, varios usuarios pueden compartir una aplicación, pero cada usuario tiene su propia configuración de dicha aplicación en su directorio en */home*. Similar al directorio “Archivos y programas” de Windows
/proc	Contiene información sobre los procesos y aplicaciones que se están ejecutando en un momento determinado en el sistema
/root	Directorio raíz del usuario root. Funciona como los directorios en */home*, pero en este caso, es solo para el superusuario (administrador del sistema)
/sbin	Comandos y programas exclusivos del superusuario (root), por ejemplo, init, route, ifup. Un usuario puede ejecutar alguna de estas aplicaciones de comandos, si tiene los permisos suficientes, o bien, si tiene la contraseña del superusuario
/srv	Almacena archivos y directorios relacionados con los servidores instalados en el sistema, tal como servidores web, de correo, etc.

/sys	Evolución de */proc*. Contiene información del sistema y de sus procesos. Este directorio utiliza un sistema de archivos virtual llamado sysfs
/tmp	Archivos temporales. En este directorio se almacenan los archivos temporales, como por ejemplo, los que guardan los navegadores de internet
/usr	Contiene todos los programas para usuarios (*/usr/bin*), bibliotecas (*/usr/lib*), documentación (*/usr/share/doc*), etc. Ésta es la parte del sistema de archivos que requiere mayor espacio. Se debe aumentar el tamaño de esta partición en función del número y tipo de paquetes que se vayan a instalar
/var	En este directorio se almacenan varios archivos con información del sistema operativo y de sus programas, tal como archivos de logs, correo electrónico de los usuarios, bases de datos, información almacenada en caché, etc.

Linux permite obtener un listado de los archivos de casi cualquier directorio (excepto de aquellos que no tienen permiso para ser accedidos).

ls muestra un listado con los archivos y directorios de un determinado directorio. Lista en orden alfabético

Sintaxis:

```
ls [-opciones] directorio1 directorio2 …
```

Opciones:

- **-a** muestra todos los archivos, incluso los ocultos
- **-l** muestra un listado en formato extendido (long) de algunos atributos de los archivos, como permisos, número de enlaces, usuario, grupo, tamaño y fecha de última modificación, además del nombre del archivo
- **-h** muestra el tamaño de los archivos
- **-d** muestra solamente el nombre del subdirectorio
- **-1** muestra el listado en una sola columna. Por defecto, el listado se muestra en varias columnas
- **-i** muestra el número del i-nodo antes del nombre de archivo
- **-R** listado recursivo. Lista primero los archivos del directorio activo, luego los de los subdirectorios de éste, y así sucesivamente
- **--color** muestra cada tipo de archivo de un color distinto, distinguiendo mediante colores, los archivos normales, los enlaces simbólicos, sockets, etc.

Ejemplo 1

```
$ ls /

bin    lib     etc      dev
usr    var     tmp      mnt
home   proc    sbin
```

La ejecución de este comando nos muestra el contenido del directorio raíz que mencionamos anteriormente.

Ejemplo 2

```
$ ls /usr

bin X11R6 includelib
localgamessbin            share
```

Muestra los archivos que contiene el directorio */usr*

Como vemos el directorio */usr* contiene a su vez otro árbol de directorios propio donde:

/usr/bin	Contiene los comandos para todos los usuarios. Son de solo lectura, pero cada usuario puede tener su propia configuración en su directorio de conexión o de login
/usr/X11R6	Contiene todos los archivos del Sistema X Window, Versión 11, Release 6.
/usr/include	Contiene las cabeceras de las librerías para lenguajes de programación
/usr/lib	Contiene bibliotecas compartidas de los binarios almacenados en */usr/bin.* Permite que las aplicaciones compartan librerías, ahorrando memoria.
/usr/local	Almacena el software instalado localmente por el administrador del sistema
/usr/games	Juegos
/usr/sbin	contiene los comandos para administrar el sistema, como por ejemplo, los programas que brindan servicios

La estructura jerárquica que posee el sistema de archivos tiene la ventaja que permite organizar los archivos en grupos comunes lógicamente o de naturaleza similar. Los archivos relacionados entre sí, residen juntos en el mismo directorio.

Directorio de conexión o Home Directory

Cuando un usuario inicia sesión, tiene un directorio de inicio de sesión con el nombre de usuario, que se encuentra debajo de */home.* Por ejemplo, si tenemos los siguientes usuarios: luz, tiago y cecilia, por debajo de */home* tendremos 3 subdirectorios de esta forma:

```
/home/luz
/home/tiago
/home/cecilia
```

Desde el punto de vista de la navegación por el sistema de archivos, estos directorios son idénticos al resto, pero son importantes porque al iniciar su sesión, cada usuario se ubica en ese punto del sistema de archivo, luego el usuario podrá desplazarse por los demás directorios, de acuerdo a los permisos que posea. El directorio de conexión se guarda en una variable de entorno llamada HOME. Cada usuario posee su propia variable HOME.

Directorio de trabajo

Un ***directorio de trabajo*** es el directorio que tiene activo un proceso en un momento determinado, es un atributo de un proceso, no de una persona ni de un programa.

pwd	muestra la ruta absoluta al directorio activo (de trabajo). Su significado es mostrar el directorio de trabajo (print working directory)

Ejemplo

```
$ pwd

/home/cecilia/
```

Indica que el directorio activo es */home/cecilia*

cd	permite cambiar el directorio activo (de trabajo). Su significado es cambiar de directorio (change directory)

Sintaxis:

```
cd nombre-directorio
```

Ejemplo 1

```
$ cd  /etc
```

Ahora el directorio es activo es */etc*. Podemos verificarlo con el comando pwd.

Ejemplo 2

```
$ cd  o  cd ~
```

Cambia al directorio de conexión del usuario, **~** hace referencia al directorio de conexión. Siguiendo el ejemplo anterior seria */home/cecilia*.

Ejemplo 3

```
$ cd ~/practicos
```

Cambia al directorio */home/cecilia/practicos* porque **~** hace referencia al directorio de conexión del usuario.

Trayectorias o rutas de acceso

Una trayectoria o ruta de acceso no es más que un camino capaz de designar la ubicación de un archivo o directorio en el que estamos trabajando. Se pueden clasificar en:

- rutas absolutas
- rutas relativas

Una **ruta absoluta** comienza siempre en el directorio raíz (comenzarán con el símbolo "*/*") y continúa con los directorios que hay que recorrer hasta llegar al archivo buscado, colocando el símbolo "*/*" entre los componentes de la ruta.

Ejemplo 1

```
$ ls  /etc/rc.d
```

Lista el contenido del directorio */etc/rc.d*

Ejemplo 2

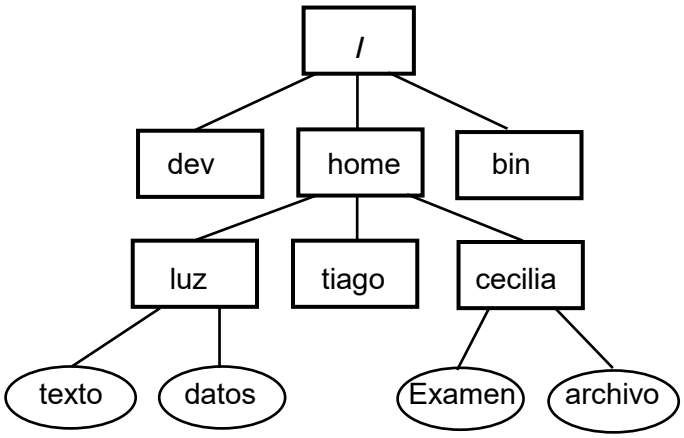
```
$ ls /home/cecilia/practico/tp-sop
```

Lista el directorio */home/cecilia/practico/tp-sop*

El uso habitual de estas rutas de acceso absolutas hace que el trabajo sea bastante tedioso, por lo que surge el concepto de ruta relativa.

Una **ruta relativa** utiliza el directorio de trabajo como punto de partida para especificar el archivo o directorio. Se vale de los directorios especiales ***(.)*** y ***(..)***. Cuando se crea un directorio, dentro de éste se crean automáticamente dos directorios, el directorio **punto *(.) ,*** que representa al directorio creado y el directorio **punto-punto *(..)*** que representa al directorio padre del directorio creado, es decir aquel que está en un nivel superior en el árbol de directorios. Ambos pueden usarse para designar a sus directorios equivalentes al formar parte de una trayectoria.

Veamos un ejemplo en el siguiente gráfico:



Para referirnos al archivo *Examen* utilizamos la siguiente trayectoria absoluta: */home/cecilia/Examen.* Para utilizar la trayectoria relativa debemos tener en cuenta nuestro *directorio de trabajo*.

Si el **directorio de trabajo es *cecilia*** *la ruta es*: *Examen ó* ***./****Examen*

Si el **directorio de trabajo es *tiago*** la ruta es: ***..****/cecilia/Examen*

Donde indicamos el directorio padre de nuestro directorio de trabajo (home) con **..** y luego descendemos en la jerarquía al directorio (*cecilia*) donde se encuentra el archivo *Examen*

Cuando ejecutamos un comando sólo con su nombre, podríamos pensar que el archivo ejecutable está en nuestro directorio de trabajo. Sin embargo, el sistema busca el archivo en la lista de directorios que están definidos como rutas de acceso predeterminadas (PATH). Este tema se ampliará en el capitulo que trata sobre Variables del shell.

Ejemplo 3

Suponemos que el directorio activo es */home/cecilia*

```
$ ls - l  /
```

Y obtenemos lo siguiente:

```
drwxr-xr-x  2  root  root      4096   May     10       2002      bin
drwxr-xr-x  3  root  root      1024   May     10       2002      boot
drwxr-xr-x  7  root  root     36864   May     10       2002      dev
drwxr-xr-x  43 root  root      4096   May     10       2002      etc
drwxr-xr-x  6  root  root      4096   Feb      6       1996      home
```

Ejemplo 4

Para ver el contenido de un directorio en particular, podemos ejecutar el comando *ls* (desde nuestro directorio de trabajo) colocando la trayectoria del directorio deseado como argumento o bien cambiarnos de directorio con el comando *cd* y luego ejecutar *ls*.

Si queremos listar */bin*:

```
ls /bin            # utilizamos ruta absoluta
ls ../../bin       # utilizamos ruta relativa desde /home/cecilia
ó
cd /bin            # cambiamos el directorio activo a /bin y luego
ls                   ejecutamos ls (en donde el argumento es . que
                     puede o no estar visible.
```

Actividad 1

1. ¿Cuál es el tipo de Sistema de Archivo en Linux, y cuál es la estructura que utiliza?
2. En cuál de los siguientes directorios se encuentran los comandos los comandos básicos de ejecución del sistema:

   a. /bin b. /dev c. /etc d. /lib e. Ninguna
3. En Linux, los periféricos se manejan a través de archivos especiales ubicados en:

   a. /bin b. /dev c. /etc d. /lib e. Ninguna
4. Analice la diferencia entre el directorio de trabajo y el directorio de conexión. ¿Deben ser siempre diferentes? Justifique su respuesta.

Actividad 2

1. Mostrar el contenido del directorio raíz. Identifique algunos de sus directorios.
   ```
   ls /
   ```
2. Igual al anterior pero acceder con camino relativo.
   ```
   ls ../..
   ```
3. Mostrar el contenido del directorio */dev*
   ```
   ls /dev
   ```

/dev es el directorio de los dispositivos. Los archivos en */dev* son los archivos de dispositivos especiales para todos los dispositivos de hardware. Contendrá un archivo por cada dispositivo que el kernel de Linux puede soportar. También contiene un script llamado MAKEDEV el cual puede crear dispositivos cuando se necesiten.

4. Muestre nuevamente el contenido de */dev*, pero utilice la trayectoria relativa desde su directorio de trabajo.
   ```
   ls ../../dev      # suponemos que el directorio activo es su
                       directorio de conexión.
   ```

5. Investigue los tipos de dispositivos que posee su sistema actualmente, ubicados en el directorio */dev*.

6. Cambiar al directorio */etc* y mostrar su contenido en forma extendida

   ```
   cd /etc
   ls – l
   ```

/etc contiene archivos y directorios de configuración de sistema local de máquina. Algunos de ellos son:
X11: Archivos de configuración para el x11. Este directorio es necesario para permitir el control local si /usr se monta sólo-lectura. Los archivos que deben ir en este directorio incluyen Xconfig (y/o XF86Config) y Xmodmap.
skel: Esqueletos de configuración de usuarios, que le son dados por defecto cuando un nuevo usuario recibe una cuenta.

7. Teclear en su terminal el comando necesario para determinar cuál es su directorio de trabajo y escriba la respuesta. _____

8. ¿Es el directorio de inicio de sesión? _____

9. El archivo *cat* (manual del comando de igual nombre) se encuentra en el subdirectorio *man1* dentro del directorio *man* que a su vez depende /usr, cual es la trayectoria absoluta para llegar a éste archivo?_____

10. Cuál es la ruta relativa si su directorio de trabajo es */usr/bin*? _____

Actividad 3

1. Dar 4 trayectorias distintas al archivo */etc/group*. Indique en cada caso el directorio de trabajo.

2. Mencionar qué archivos contienen los directorios */bin* y */sbin*. Explique las diferencias.

3. Mostrar el contenido del directorio donde Linux archiva los drivers.

4. Acceder al directorio */boot* y analizar su contenido.

5. La posibilidad que brinda el sistema de archivo de nombrar a más de un archivo con el mismo nombre simbólico (por ejemplo *proc-linux*) esta dada por:

a) la tabla de asignación de archivos
b) la estructura jerárquica de archivos
c) los archivos **.** y **..**
d) los Runlevel

Capítulo 4

Manejo de Archivos

Desde el punto de vista del sistema, un archivo no es más que una secuencia de bytes identificado con un número i-nodo que permite su localización en disco.

Desde el punto de vista del usuario, los archivos existen para guardar información y luego poder recuperarla. El usuario les otorga a éstos un nombre simbólico para poder designarlos. Linux dispone de un numeroso conjunto de utilidades para manipular archivos (individualmente o en grupos). A través de esta modalidad de trabajo es posible alcanzar un alto nivel de flexibilidad y versatilidad para realizar operaciones de cualquier grado de complejidad.

En este capítulo veremos cómo utilizar los comandos básicos para la manipulación de archivos y directorios, por ejemplo crear y borrar, mostrar el contenido, copiar y mover archivos dentro del Sistema de Archivos y además cómo Linux permite compartir los archivos. Para tratar estos temas también introduciremos el concepto de redireccionamiento que es de gran utilidad para el uso eficiente de los comandos.

Archivos

La base del sistema de archivos de Linux es obviamente el archivo, que no es otra cosa que la estructura empleada por el sistema operativo para almacenar información en un dispositivo físico como un disco duro, un DVD, un BD (Disco Blu-ray), una memoria flash, o almacenamiento de datos basado en redes como la nube (cloud) o servicios de almacenamiento online. Como es natural un archivo puede contener cualquier tipo de información. Todos los archivos de Linux tienen un nombre, el cual debe cumplir ciertas reglas:

- Un nombre de archivo puede tener entre 1 y 255 caracteres.
- Se puede utilizar cualquier caracter excepto la barra inclinada **/** y el carácter nulo, ***no*** es recomendable emplear los caracteres con significado especial en Linux, que son los siguientes: **= \ ^ ~ ' " ` * ; - ? [ ] ( ) !& ~ <>.** Para emplear archivos con estos caracteres o espacios hay que introducir el nombre del archivo entre comillas.
- Se pueden utilizar números exclusivamente si así se desea. Cabe aclarar que Linux es sensible a las letras mayúsculas y minúsculas por lo tanto se consideran diferentes, y no es lo mismo el nombre archivo *nota.txt* que *Nota.txt* ó *nota.Txt.*

En LINUX existen los siguientes tipos de archivos:

Regulares	son aquellos que pueden contener cualquier tipo de información, desde una imagen, un texto, un sonido, etc.
Directorios	(o carpetas), es un archivo especial que agrupa otros archivos de una forma estructurada.
Especiales	son la base sobre la que se asienta *Linux*, puesto que representan los dispositivos o periféricos conectados a una computadora, como puede ser una impresora. De esta forma introducir información en ese archivo equivale a enviar información a la impresora. Para el usuario estos dispositivos tienen el mismo aspecto y uso que los archivos regulares u ordinarios.
Enlazados	los enlaces le permiten dar a un único archivo múltiples nombres.
Tubería con nombre	(named pipeline). Son archivos utilizados para comunicación entre procesos.
Socket	permite que dos procesos distintos referencien y abran el mismo socket con el fin de comunicarse.

Estos tipos de archivos se identifican con los siguientes símbolos:

- archivo común, regular u ordinario
d directorio
b archivo especial de bloque
c archivo especial de carácter
l archivo enlazado
p archivo de tubería con nombre
s archivo de socket

Tabla 1. Tipos de archivos

Bloque Indice (i-nodo)

Cuando se crea un archivo, el sistema operativo agrega una entrada en la tabla de directorio donde estará ubicado dicho archivo. Cada entrada de un directorio en Linux contiene el nombre del archivo y un número llamado i-nodo. Este número es un bloque de disco que contiene toda la información necesaria para que el sistema pueda administrar el archivo. Cada archivo o directorio tiene su propio i-nodo identificado de manera única en un sistema en particular. El i-nodo contiene los atributos del archivo o directorio y los punteros que permiten localizar en orden cada uno de los bloques del archivo almacenados en el disco. A continuación se observa una entrada de directorio:

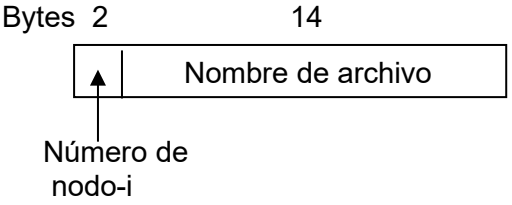


Fig. 1. Una entrada de directorio en Linux [Tanenbaum]

Podemos visualizar la tabla de directorio con la opción –i del comando *ls*, de la siguiente manera:

```
ls  -i  /
45689    bin
45692    boot
   :       :
   :       :
```

Un i-nodo contiene los atributos del archivo, las direcciones de los primeros bloques de datos y tres punteros, uno simplemente indirecto, otro doblemente indirecto y un tercer puntero denominado triplemente indirecto. A continuación se detalla la estructura de un i-nodo:

Atributos del archivo:

UID	*User identifier*. El *identificador de usuario* del creador o un propietario del archivo. Es un número que le asigna el sistema operativo a cada usuario cuando éste se registra en el sistema. Cada usuario para poder trabajar debe registrarse en el sistema. Cada archivo tiene un propietario, que es el usuario que creó el archivo.
GID	*Group identifier*. El *identificador de grupo* asociado al archivo. El GID es un número que el sistema operativo le asigna a un grupo de usuarios. Todo usuario pertenece por lo menos a un grupo de usuarios, normalmente definido por el administrador del sistema. También cada archivo “pertenece” a un grupo. El *administrador del sistema* es un usuario que tiene una jerarquía especial que le permite realizar operaciones que normalmente no están permitidas a usuarios normales. Dicho administrador se llama *root* en el sistema operativo LINUX.

Tipo de archivo	Según tabla 1. Cuando el i-nodo no tiene asignado ningún archivo, el tipo se define como *tipo 0*. De esa forma se reconoce cuando un i-nodo está *libre*.
Modo de acceso	Permisos de leer, escribir, y ejecutar el archivo.
Marcas de tiempo	fecha de última modificación (*mtime*), acceso (atime) y de alteración del propio i-nodo (ctime).
Número de enlaces	(*contador de enlaces*) el número de nombres (entradas de directorio) asociados con este i-nodo. El número de enlaces se utiliza para que el sistema operativo sepa en qué momento eliminar el contenido y el i-nodo del archivo en disco. El número de enlaces va disminuyendo cada vez que se borra un enlace a un archivo determinado. Cuando el contador llega a cero, significa que ese archivo no es referenciado desde ningún directorio, por lo cual el sistema operativo lo elimina físicamente de disco.
Longitud	Cantidad de bytes que ocupa el archivo.
Punteros a bloques	Son números de 32 bits que permiten localizar un bloque físico del disco asignado al archivo. Son los punteros *directo, simplemente indirecto, doblemente indirecto y triplemente indirecto.*

Ejemplo

Algunos de los atributos de un archivo se pueden visualizar con el comando *ls –l* (listado extendido)

```
$ ls -l  /
drwxr-xr--   2    root  root   4096    may  2 13:42      bin
drwxr-xr-x   3    root  root   1024    may  2 13:43      boo

:                     :             :
```

Este comando muestra en formato largo o extendido la información de los archivos del directorio raíz. Estos son algunos de los atributos del archivo, de izquierda a derecha:

Tipo de archivo	(ver Tabla 1)
Permisos	son nueve caracteres, donde la primer terna *rwx* (lectura, escritura y ejecución) pertenecen al dueño del archivo, la segunda terna son los permisos del grupo y la tercera terna son para los otros usuarios.
Enlace	este campo es numérico e indica la cantidad de enlaces que tiene el archivo.
Nombre del dueño del archivo	*username*
Nombre del grupo	*groupname*
Tamaño del archivo	por defecto esta información se da en bytes.
Fecha y hora	de modificación del archivo

Redireccionamiento de E/S

Cada vez que se ejecuta un shell, se abren los archivos *stdin, stdout* y *stderr.* El archivo *stdin* está asociado al teclado, tiene **0** como descriptor de archivo y representa a la entrada estándar de cualquier comando. Por ejemplo, los comandos *cat, write* y *mail* permiten que el usuario introduzca por teclado las líneas de datos, y esperan a que se les indique con <ctrl.d > que la carga ha finalizado. Estos comandos toman la entrada de datos del archivo *stdin.*

Mediante la redirección de entrada se puede indicar a un comando que tome como entrada los datos de un archivo determinado distinto a *stdin*, utilizando el símbolo **<** y el nombre del archivo de entrada.

Ejemplo 1

```
$ mail nom-usuario < mensaje
```

Donde *nom-usuario* es el nombre del destinatario de los datos contenidos en el archivo *mensaje.* La redirección de entrada no produce modificaciones en el archivo que se utiliza como entrada.

Los comandos envían su salida por defecto a la salida estándar, la pantalla, es decir al archivo *stdout* asociado a ella. El descriptor de este archivo es 1. La redirección de salida permite recoger la información que presenta un comando en pantalla y almacenarla en un archivo. Para indicar la redirección de salida se utilizan los símbolos **>** y **>>**.

Al utilizar el símbolo **>** el archivo de salida se sobrescribe con los nuevos datos; en el caso que el archivo no exista, se crea con la misma orden.

Ejemplo 2

```
$ ls / > dire
```

El comando *ls* lista el contenido del directorio raíz, sin embargo la salida no aparecerá por pantalla sino que se redirecciona al archivo *dire*, que si no existe, se crea en la misma operación.

Al utilizar **>>**, redireccionamos la salida a un archivo determinado y si éste no existe se crea. Si el archivo existe, los nuevos datos redireccionados se añaden al final del archivo sin dañar las líneas del archivo que ya existían.

Ejemplo 3

```
$ ls  /dev  >>  dire
```

Con el símbolo >>, agregamos al final del archivo *dire* un listado de los archivos del directorio /*dev.*

S*tderr* es un archivo estándar que recibe los mensajes de error, tiene asociado el descriptor 2. Cuando el shell no puede ejecutar un comando muestra un mensaje de error por la pantalla, si necesitamos redireccionar los mensajes de error tendremos que insertar en la línea de comandos el descriptor 2 del archivo stderr acompañado por **>** o **>>**, como sigue:

Ejemplo 4

```
$ cp   2>  fichero.err
```

Como al comando cp no le hemos pasado como argumento el nombre del archivo a copiar, generará un mensaje de error que en vez de mostrarse por pantalla, se redirecciona al archivo *fichero.err*. Si visualizamos el archivo creado *fichero.err* obtendremos la siguiente salida:

Ejemplo 5

```
$ cat fichero.err
cp : faltan argumentos (ficheros)
pruebe cp – help para más información
```

Manejo de Archivos

Para crear archivos de texto siempre es posible utilizar algún editor como vi, gedit, ed, etc. Pero también podemos usar comandos.

touch	crea un archivo de texto vacío.

Sintaxis:

```
touch  nombre-archivo
```

cat	muestra, concatena y permite crear archivos.

Sintaxis:

```
cat nombre-archivo
```

La forma más sencilla de crear un archivo es utilizando el comando *cat*, que también nos permite ver el contenido de un archivo y concatenar archivos, pero lo que nos interesa es que *cat* junto con el concepto de redireccionamiento, nos permite crear archivos de la siguiente forma:

Ejemplo 1

```
$ cat > materias
```

El comando *cat* toma datos de la entrada estándar (el teclado) y los redirecciona (los guarda) en el archivo *materias*. En este caso el usuario puede tipear todas las líneas necesarias y justificarlas con la tecla <Enter>, pero una vez justificadas, no podrá regresar a corregir lo escrito. Para grabar los datos y regresar al shell pulse ctrl.d. Deberemos ingresar:

```
> Sistemas Operativos
> Química
> Estadística
ctrl.. d
```

El concepto de redireccionamiento es muy importante pues nos permite crear un archivo con la salida que genere cualquier comando, provocando que la salida de éste ya no sea enviada a la salida estándar sino a un determinado archivo. Así por ejemplo podríamos querer crear un archivo con todos los comandos que están en */bin*:

Ejemplo 2

```
$ ls /bin > comandos
$ ls
materias    comandos
```

El comando *ls* debe mostrar por pantalla un listado de los archivos del directorio */bin*. Sin embargo con la línea de comando anterior, el shell devuelve sólo el prompt. Pero si ejecuta la orden *ls* podrá ver al archivo *comandos* formando parte del listado.

La forma más sencilla de ver un archivo es con el comando *cat*. Muestra el contenido de todos los archivos que se le pasen como argumento.

Ejemplo 3

```
$ cat datos practico
```

Es posible también, con este mismo comando concatenar (unir) archivos, simplemente redireccionando la salida secuencial del ejemplo anterior a un tercer archivo.

Ejemplo 4

```
$ cat  datos  practico  >  salida
```

Esto no produce modificaciones en los archivos tomados como argumento. El comando anterior, crea el archivo salida, cuyo contenido está formado por lo que contienen los archivos datos y practico.

mkdir	crea un directorio o lista de directorios bajo la ruta de acceso indicada.

Sintaxis:

```
mkdir  directorio-nuevo
```

Crea el directorio *directorio-nuevo* por debajo del directorio actual o de trabajo.

Opción:

-p permite crear los directorios padres, en caso que no existan.

Si queremos crear un directorio, como subdirectorio de otro específico, debemos indicar la ruta absoluta o relativa del directorio nuevo.

Ejemplo

```
$ mkdir  /home/ana/directorio-nuevo
```

rm	borra un archivo o una lista de archivos.

Sintaxis:

```
rm  nom-archivo
```

Opciones

- **-i** pide al usuario que confirme la eliminación del archivo.
- **-f** borra el archivo sin tener en cuenta los permisos del archivo.
- **-r** se utiliza para borrar el contenido de un directorio en forma recursiva, incluso borra el directorio dado como argumento.
- **-v** explica qué se está haciendo.

Al ejecutar este comando, borramos en la tabla del directorio la entrada correspondiente a *nom-archivo.* El argumento puede incluir una lista de archivos a borrar o puede también utilizar metacaracteres (consultar el Apéndice 1).

Ejemplo 1

```
$ rm materias comandos
```

Este comando borra dos archivos, *materias y comandos*, que están en el directorio actual.

Sin embargo podemos eliminar un directorio y subdirectorios con el comando *rm –r*, pues borra en forma recursiva desde el directorio que se dio como argumento. Supongamos que *listados* es un directorio que necesitamos borrar, pero no está vacío.

Ejemplo 2

```
$ rm -r listados
```

Este comando borra todos los archivos y subdirectorios que contiene *listados*, incluso borra el propio directorio *listados*.

rmdir	borra un directorio que este vacío.

Sintaxis:

```
rmdir directorio-nuevo
```

Para poder eliminar un directorio, éste debe estar vacío. Por lo tanto la tarea previa a la eliminación de cualquier directorio, es eliminar todos los archivos que éste contiene.

more	muestra un archivo en forma paginada.

Sintaxis:

```
more [-n° de líneas p/página]  nom-archivos
```

Opciones:

-n cantidad de líneas que muestra por página.

Uno de los problemas que presenta la utilización del comando *cat* es que produce la visualización completa del archivo, sin pausas, por lo que si éste es demasiado largo sólo podremos ver las últimas 20 líneas en la pantalla. El comando more nos permite ver el contenido de un archivo, página por página. Por defecto en cada página muestra 21 líneas.

Ejemplo

```
$ more -15  datos
```

Por supuesto también al comando more podemos darle como argumento el nombre de más de un archivo, y los mostrará en forma secuencial.

Una vez que el comando *more* nos está mostrando el contenido de un archivo, tenemos opción para ejecutar algunos comandos:

- barra espaciadora avanza a la página siguiente.
- tecla q cierra el programa more.
- tecla s ó <Enter> avanza de a una línea.
- tecla f avanza toda una página del archivo.

- /<patrón> busca y avanza hasta una palabra determinada.
- tecla b: retrocede una página.
- tecla ? ó h: muestra la ayuda para el comando *more*.

head muestra las n primeras líneas de un archivo.

Sintaxis:

```
head [-n° de líneas] nom-archivo
```

El comando *head* nos permite visualizar por pantalla las primeras líneas de un archivo. Por ejemplo, podríamos querer ver la cabecera de algún mail que nos enviaron o saber quienes son los primeros usuarios en conectarse al sistema.

Opciones:

-n muesta las n primeras líneas del archivo.
-c muestra los primeros n bytes del archivo.
- - bytes [ ]k muestra los primeros n kbytes del archivo.

La opción *n* es la cantidad de líneas que necesitamos ver, si se omite la opción por defecto muestra las primeras 10 líneas

Ejemplo

```
$ head -5  accesos
jpsuarez
lcgarcía
iptorrez
fbalvarez
mmsolver
```

Suponga que el archivo *acceso* almacena información de los usuarios que se han conectado al sistema en las últimas 24 horas. Cada línea del archivo tiene información de un usuario. El comando anterior mostrará las primeras 5 líneas del archivo, es decir información de los primeros 5 usuarios conectados.

tail muestra las últimas líneas de un archivo.

Sintaxis:

```
tail [-n° de líneas] nom-archivo
```

El comando *tail* muestra las últimas líneas de un archivo, por defecto muestras las diez últimas líneas. Este comando es muy útil porque los archivos crecen, y lo hacen por la parte inferior. Es decir que se le añaden registros, si no tiene un orden predeterminado, en la parte inferior del archivo. Un usuario puede querer ver sólo los últimos elementos añadidos y no todo el archivo.

Opciones:

-n muestra las n primeras líneas del archivo.
-c muestra los primeros n bytes del archivo.
-f muestra las líneas que se van adicionando a un archivo.

Este comando también tiene la capacidad de mostrar los archivos mientras crecen, con la opción -f el comando *tail* abre el archivo, muestra las últimas líneas y comienza a supervisarlo. Cada vez que se introduzca una nueva línea al archivo, ésta aparecerá directamente en la pantalla del usuario.

cp	copia un archivo/lista de archivos en un directorio.

Copiar un archivo implica generar otro archivo, con distinto número de i-nodo pero con el mismo contenido del archivo anterior. Si ubicamos la copia en un directorio distinto al que contiene al archivo, podremos utilizar el mismo nombre. Pero si ubicamos la copia en el mismo directorio, debemos utilizar un nombre diferente, pues sino el sistema nos dará un mensaje de error, como el siguiente:

```
El archivo ya existe, desea sobreescribirlo?
```

Para realizar copias de archivos debemos utilizar el comando *cp*, que nos permite realizar varias operaciones:

- Copiar un archivo a otro:

```
cp arch-origen arch-destino
```

Donde tanto en *arch-origen* como *arch-destino* debemos incluir la ruta de acceso al archivo, sea esta absoluta o relativa.

- copiar un conjunto de archivos a un directorio específico:

```
cp archivo-1 archivo-2 archivo-n direc-destino
```

- Copiar en forma recursiva un directorio completo, para ello debemos utilizar la opción -r:

```
cp -r direc-origen direc-destino
```

Ejemplo 1

```
$ cp listado  listacop
```

Este comando copia el archivo *listado* al archivo *listacop*, ambos archivos están en el directorio actual. Genera una copia con los mismos datos y con los mismos atributos pero con distinto número de i-nodo, es decir, que a partir de ahora los archivos *listado* y *listacop,* evolucionarán independientemente. Para comprobar lo que hemos dicho, tipee el siguiente comando:

```
$ ls –li
```

Ejemplo 2

```
$ cp listado   ../
```

Este comando copia el archivo *listado,* que está en el directorio actual, al directorio padre con el mismo nombre *listado***.**

Ejemplo 3

```
$ cp  listado  listacop  /home/rodriquez/
```

Este comando copia los archivos *listado y listacop* al directorio *rodriguez,* que está en */home*.

Ejemplo 4

```
$ cp -r /home/rodríguez/practicos /tmp
```

Este comando crea un directorio *practicos* en el directorio */tmp*, y realiza una copia de todos los archivos y directorios de /home/rodríguez/practico a /tmp/practico.

mv	mueve o renombra archivos.

El renombrar un archivo implica cambiarle el nombre no su i-nodo. Si la operación consiste en mover el archivo a otro directorio, implica simplemente cambiar de tabla de directorio el archivo conservando el número de i-nodo y por supuesto sus atributos.

Sintaxis:

```
mv nom-archivo nom-nuevo
```

El comando *mv* puede mover o desplazar un conjunto de archivos de un directorio a otro.

Sintaxis:

```
mv archivo-1 archivo-2 ...  direc-destino
```

Ejemplo 1

```
$ mv listado listacop
```

Cambia el nombre del archivo *listado* por *listacop*, dentro de un mismo directorio.

Ejemplo 2

```
$ mv  listado lista  /home/suarez/
```

Este comando borra los archivos *listado* y *lista* de la tabla de directorio actual y los agrega a la tabla de directorio *suarez.* Como está modificando directorios, necesita en ellos el permiso de escritura (w).

Archivos compartidos - Enlaces

Existen ocasiones en donde varios usuarios necesitan compartir los archivos. Conviene que un archivo compartido aparezca en forma simultánea en distintos directorios los cuales pertenecen a distintos usuarios. Un archivo compartido es muy conveniente pero puede ser muy problemático. Linux enfrenta esta situación a través de lo que llamamos enlaces.

Los enlaces le permiten dar a un único archivo múltiples nombres, pero los archivos son identificados por el sistema por su número de i-nodo, el cual es el único identificador del archivo para el sistema de archivos.

Un directorio es una lista de números de i-nodo con sus correspondientes nombres de archivo. Cada nombre de archivo en un directorio es un enlace a un i-nodo particular.

ln crea enlaces entre archivos.

Sintaxis:

```
ln [opción] archivo1 nombre-del-enlace
```

Opciones:

- **-d** Permite al *super-usuario* hacer enlaces duros a directorios.
- **-s** Crear enlace simbólico.
- **-f** Borrar los archivos de destino que ya existen.

Enlaces duros

Al crear un enlace duro se crea un nuevo acceso al archivo desde el mismo directorio en donde se encuentre el archivo enlazado o en otro punto del sistema de archivos. Así, un mismo archivo puede aparecer en varios sitios del sistema de archivos simultáneamente, pero ocupar un único espacio en disco. Realmente sólo hay un archivo físico, los demás son *enlaces duros* o *hard links*. Esto es posible gracias a que Linux separa la información de un archivo de su entrada en un directorio, permitiendo así tener varias entradas de directorio para un único archivo.

Cada vez que se crea un enlace a un archivo, se incrementa el contador de enlaces. Es decir que este número nos indica desde cuántos puntos distintos del file system puede ser accedido el archivo. El archivo no desaparece físicamente del disco hasta que no se borren todos sus enlaces, es decir hasta que el contador de enlaces sea 0.

Los enlaces duros tienen la limitación de que deben estar en el mismo volumen físico. Esta limitación se supera en Linux con los enlaces simbólicos.

Ejemplo

```
$ ln notas calificaciones
```

Con ls -i veremos que los dos archivos tienen el mismo número de i-nodo.

```
$ ls –i notas calificaciones
46192 notas    46192   calificaciones
```

Si se realizan cambios en el archivo *notas*, estos cambios también serán efectuados en el archivo *calificaciones*. Para todos los efectos, *notas* y *calificaciones* son el mismo archivo.

Estos enlaces son conocidos como enlaces duros (hard links) porque directamente crean el enlace al i-nodo. Solo podemos crear enlaces duros entre archivos del mismo sistema de archivos; los enlaces simbólicos no tienen esta restricción.

Enlaces simbólicos

Al enlazar en forma simbólica un archivo, se crea un nuevo archivo tipo l (link) con un nuevo número de i-nodo. Desde éste, se puede acceder al archivo enlazado mediante su ruta absoluta. Dato que está almacenado en el archivo tipo link.

Ejemplo

```
$  ln –s notas evaluaciones
```

Usando ls -i, veremos que los dos archivos tienen número de i-nodos diferente:

```
$ ls –i notas evaluaciones
  46192 notas    45156   evaluaciones
```

Si ejecutamos el comando *ls -l* vemos que el archivo *evaluaciones* es un enlace simbólico apuntando a *notas.* También podemos ver el número de enlaces de un archivo.

```
$ ls –l notas evaluaciones
lrwxrwxrwx  1 root   root  3   Aug  5   16:52  evaluaciones -> notas
-rw-r--r--  1 root   root  12  Aug  5   16:50  notas
```

Funcionalmente, los enlaces duros y simbólicos son similares, pero hay algunas diferencias:

- Todas las acciones realizadas sobre un enlace simbólico se realizan realmente sobre el archivo original, salvo los comandos */s* y *rm.*
- Los bits de permisos en un enlace simbólico no se usan (siempre aparecen como rwxrwxrwx). En su lugar, los permisos del enlace simbólico son determinados por los permisos del archivo "apuntado" por el enlace.
- Si el archivo original se borrara, el enlace simbólico apuntará a un archivo que ya no existe. En cambio, si se borra un enlace duro a un archivo, lo único que se modifica es el atributo número de enlace, pero el archivo original sigue existiendo. Sólo desaparece físicamente del disco el archivo original, cuando se borra el último enlace.
- Los enlaces duros tienen la desventaja que registran como propietario (en el i-nodo) al usuario creador del archivo. La creación de un enlace a ese archivo no modifica la propiedad. Si el propietario desea eliminar el archivo, no puede hacerlo hasta que el segundo usuario decida borrarlo y el contador de enlaces lo permita. Los enlaces simbólicos son otro archivo conteniendo un camino o *pathname*. Cuando el propietario lo elimina el archivo se destruye.
- Los enlaces simbólicos son de ayuda puesto que identifican al archivo al que apuntan; con enlaces duros no hay forma fácil de saber qué archivo está enlazado al mismo número de i-nodo. Se puede averiguar con el comando *find* y la opción *inum*.

Actividad 1

Los siguientes puntos conforman una guía resuelta de ejercicios donde el alumno deberá usar la línea de comandos (modo texto) del sistema operativo Linux.

1. Muestre un listado extendido del directorio */dev*, y analice la información.

```
ls  -l  /dev
```

Observe que en el listado extendido que le muestra ls, el primer carácter en cada una de las líneas visualizadas identifica el tipo de archivo de que se trata. En este directorio en particular, el primer carácter será *b* ó *c* identificando a los archivos de bloque ó caracter respectivamente.

2. Muestre el directorio /bin, y analice la información.

```
ls  -l  /bin
```

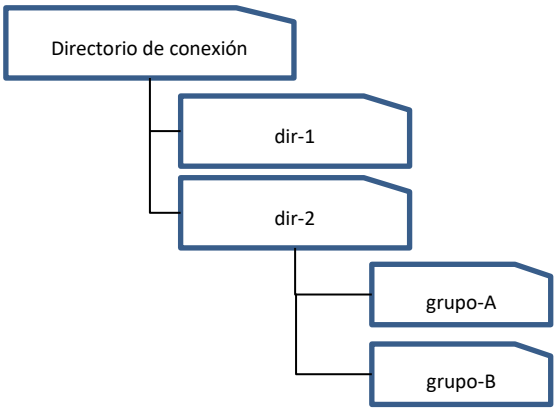
En este directorio podrá ver que algunas filas en el primer caracter tiene un guión (-) para los archivos ordinarios, o "l" para los archivos enlazados.

3. Verifique que se encuentra trabajando en el directorio de conexión, de lo contrario desplácese a ese directorio.

```
pwd
```

Si la salida obtenida no es la ruta absoluta a su directorio de conexión tipee el siguiente comando: cd

4. Construya la siguiente estructura de subdirectorios:




```
kdir dir-1
mkdir dir-2
mkdir  dir-2/grupo-A
mkdir  dir-2/grupo-B
ls -R
```

En estas líneas de comandos, ¿utilizó camino absoluto o relativo para nombrar los directorios?

5. En su directorio de login, guarde sus datos personales en el archivo *personal.*

```
cat > personal
```

En este último comando se utilizó el concepto de redireccionamiento, que combinado con el comando *cat* permite que todo lo ingresado por teclado se muestre por pantalla y luego se grabe en el archivo. Para grabar todo lo ingresado deberá teclear *ctrl d*.

6. Cámbiese al directorio grupo-A.

```
cd  dir-2/grupo-A
```

Ahora su directorio actual es grupo-A

7. Guarde el contenido del directorio *bin*(comandos del sistema) en el archivo *comandos* en su directorio de conexión.

```
ls   /bin  >   ../../comandos
```

En este ejemplo el comando *ls /bin* muestra por pantalla el contenido del directorio *bin* (se utilizó ruta absoluta para nombrarlo) pero combinado con el redireccionamiento, la salida ya no aparece por pantalla (salida estándar) sino que se guarda en el archivo *comandos*, que sino existe se crea en este momento. Entonces para ver el contenido del directorio *bin* deberemos ver el contenido del archivo.

```
cat  ../../ comandos
```

Hemos utilizado ruta relativa para nombrar el archivo porque está en el directorio de conexión.

8. Agréguele al archivo creado en el punto anterior el contenido del directorio home (ya habrá observado que es aquí donde están definidos los directorios de conexión de los usuarios).

```
ls  /home  >>  ../../comandos
```

En este caso también utilizamos redireccionamiento, pero con >>, no borramos lo que tiene el archivo sino que los agregamos al final del mismo.

9. Regrese al directorio de conexión.

```
cd
```

10. Muestre el contenido del archivo *comandos* haciendo pausa luego de cada pantalla.

```
more comandos
```

11. Especifique al comando *more* la cantidad de líneas que desea visualizar por página.

```
more  -7  comandos
```

12. Cree el archivo *partescom* de manera tal que contenga las 15 primeras líneas del archivo *comandos*. Verifique su contenido.

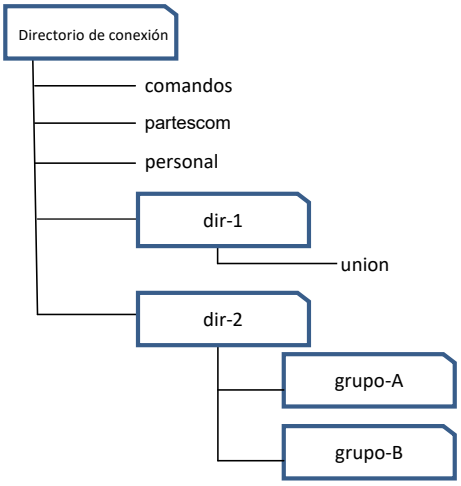
```
head  -15  comandos  >  partescom
cat  partescom
```

Observe que el archivo *partescom* tiene 15 líneas correspondientes a los primeros comandos del directorio */bin* ordenados alfabéticamente.

13. Concatene los archivos *partescom* y *comandos* y guarde dicha información en el archivo *union* en el subdirectorio *dir-1*. Cámbiese a él.

```
cat  partescom  comandos  >  dir-1/union
cd   dir_1
```

14. En este momento, el árbol de directorios y archivos debe ser el siguiente:



15. Borre los archivos y directorios creados. Cierre la sesión.

a. Primero nos ubicamos en nuestro directorio de conexión
```
cd
```
b. Luego ejecutamos el comando para borrar en forma recursiva la rama del directorio dir-2.
```
rm -r dir-2
```
c. Luego hacemos lo mismo para el directorio dir-1
```
rm -r dir-1
```
d. Por último borramos los archivos que creamos en el directorio de conexión.
```
rm comandos personal partescom
```
e. Cerramos la sesión
```
logout
```

Actividad 2

Los siguientes puntos conforman una guía resuelta de ejercicios donde el alumno deberá usa la línea de comandos (modo texto) del sistema operativo Linux.

1. Comenzar la sesión de trabajo
```
login:
password:
```

2. Crear el archivo *dato1* que contenga el nombre de los archivos del directorio /bin que comiencen con la letra **c**.
```
ls /bin/c* > dato1
```

En la ejecución de este comando se usó redireccionamiento por lo que la salida del comando *ls* no aparece por pantalla. Si ejecutamos *more dato1* vamos a ver el nombre de los archivos que están en el directorio /bin y que comienzan con la letra c. El metacaracter * (todos) reemplaza a todos los caracteres que están después de la letra **c** en los nombres de archivo del directorio /bin.

3. Copiar el archivo *dato1* como *dato2.*
```
cp dato1 dato2
```

El comando *cp* genera una copia física del archivo *dato1*, con el nombre *dato2*. A partir de aquí los archivos podrán evolucionar en forma independiente. Es decir, las modificaciones realizadas sobre uno de los archivos no se reflejarán en el otro.

4. Mostrar el número de i-nodo de los archivos utilizados en el punto anterior.
```
ls -i dato1 dato2
```

Esta opción del *ls* nos muestra el número de i-nodo de cada uno de los archivos.
Vemos que *dato1* y *dato2* tienen diferente número de i-nodo, a pesar de ser una copia uno del otro. ¿Por qué?

5. Agregar al archivo *dato2*el nombre de los archivos del directorio */dev* que comiencen con **h**.
```
ls /dev/h* >> dato2
```

El concepto utilizado en este ítem es también el redireccionamiento, sin embargo no vamos a sobrescribir la información, sino agregar al final del archivo.

6. Mostrar en formato extendido, el contenido del directorio de trabajo.

```
ls -l
```

Este comando muestra en formato extendido la información de los archivos del directorio de trabajo. Estos son algunos de los atributos del archivo, de izquierda a derecha:

Tipo de archivo / Permisos / Enlace / Nombre del dueño del archivo / Nombre del Grupo
Tamaño del archivo / Fecha y hora / Nombre del archivo

7. Mostrar todos los archivos de su directorio, incluso los ocultos.

```
ls -a
```

La opción **a** muestra todos los archivos, incluso los ocultos que se identifican porque el nombre comienza con un punto.

8. Crear un directorio bajo el nombre de *nuevo.*

```
mkdir nuevo
```

Al crear el directorio pasamos como argumento al comando, la ruta relativa del directorio. Funcionaría lo mismo si tipeamos *mkdir ./nuevo*, es decir crea el directorio dentro del directorio actual.

9. Ejecutar el siguiente comando: *ls –li* y analizar la salida obtenida.
Vemos que nos muestra el número de i-nodo y algunos de los atributos de cada archivo del directorio. A ésta información el sistema la obtiene de la tabla de nodos-i. Registre el número de i-nodo de los archivos *dato2* y *dato1*.

¿Los directorios tienen número de i-nodo? ¿Por qué? _____

Nº i-nodo de dato1: _____ Nº i-nodo de dato2: _____

10. Cambiarse al directorio *nuevo*

```
cd nuevo
```

Cambiamos de posición en el árbol de directorios.
La ejecución del comando *pwd* nos dice que la posición actual es: _____

11. Mover los archivos *dato1 y dato2*al directorio actual.

```
mv ../dato1 ../dato2 .
```

Analice el resultado del comando anterior y verá que los archivos *dato1 y dato2* ya no están en el directorio padre (home directory) sino que están en el directorio actual (representado por un punto en la línea de comandos).

12. Verifique el contenido del directorio de conexión. ¿Aparecen los archivos *dato1* y dato2?

```
ls ..
```

Vemos que los archivos no se encuentran en este directorio.

13. Verifique el contenido del directorio actual. Muestre los números de i-nodos.

```
ls -i
```

En la salida de este comando preste atención al n° de i-nodo de los archivos.
¿El i-nodo de los archivos movidos son los mismos que apuntó el item 9? Explique.

14. Realice un enlace duro del archivo *dato2* como *dato2ln* y verifique el contador de enlaces y N° de i-nodo. (Ud. está ubicado en el directorio *nuevo*)

```
ln   dato2   dato2ln
ls   -li
```

El comando *ln* crea una entrada en la tabla directorio para el archivo *dato2ln*, con el mismo N° de i-nodo . En este caso se creó en el mismo directorio pero podría haber sido cualquier otro, por ejemplo: *../dato2n*. De esta manera se puede acceder desde distintas partes del árbol de directorios a un mismo i-nodo.
El listado extendido muestra que el contador de enlaces se incrementó, ahora es 2 en ambos archivos, como también los atributos que se pueden ver (permisos, tamaño, dueño, grupo, etc) son los mismos.

15. Realice un enlace simbólico del archivo *dato2* como *dato2sb* y verifique el contador de enlaces y N° de i-nodo.

```
ln  -s dato2 dato2sb
ls  -li
```

Observe que el i-nodo y los atributos del archivo *dato2sb*, son distintos a los de *dato2,* pues es otro archivo, cuyo contenido es /*home/user-name/nuevo/dato2*, es decir que llamando a *dato2sb* accedemos a *dato2*. Con éste tipo de enlace podemos saber cuál es el archivo enlazado.

16. Borre el archivo *dato2* y verifique el contador de enlaces.

```
rm dato2
ls -i
```

El contador de enlaces del archivo *dato2ln* ha disminuido en uno, se ha borrado unlink, pero el archivo físico aún existe. El sistema lo borrará sólo cuando el contadorde enlaces sea cero.

17. Mostrar el contenido del archivo *datos2sb.*

```
cat datos2sb
```

El sistema nos envía un mensaje de error, ya que *datos2sb* es un enlace a un archivo que ya no existe.

18. Borrar los archivos creados y terminar la sesión.

```
cd
rm -r nuevo
ctrl  d
```

Actividad 3

1. El enlace duro crea una nueva entrada en el directorio, con el nº de i-nodo del archivo enlazado. **V** ó **F**

2. Al borrar un archivo, borramos un acceso o enlace al i-nodo. **V** ó **F**

3. Mostrar el nº de i-nodo de los archivos de /bin:
   a. en una columna
   b. en varias columnas.

4. Generar un enlace simbólico al archivo /etc/inittab. Explicar las características de este tipo de enlace en base al ejemplo.

5. Crear el archivo ***A*** y mostrar su contenido paginado de 5 líneas por página.

6. Mover el archivo ***A*** al directorio padre. ¿Tuvo éxito la acción anterior? Justifique.

7. Explicar en qué consiste la acción de mover un archivo a otro directorio.

Actividad 4

1. Averigüe cuál es su directorio de conexión y cree en él un directorio que se llame *clave*.

2. Copie el archivo *passwd* del sistema (su localización y sintaxis se puede encontrar utilizando el *man*) en el directorio *clave.* Realice la copia de dos modos distintos:
   a. Especificando tanto el origen como el destino mediante trayectorias absolutas.
   b. Especificando origen y destino con trayectorias relativas.

3. Utilizando el comando **ln** (Use **man ln**):
   a. Cree un enlace duro a un archivo situado en su directorio de conexión y compare el número de enlaces que tiene ahora y los que tenía antes de realizar esta operación.
   b. Realice de nuevo la misma operación pero en este caso sobre un directorio. ¿Puede realizar un enlace simbólico sobre un directorio?
   c. Cree un enlace simbólico sobre el archivo original de la pregunta a) ¿Cuántos enlaces tiene ahora?
   d. Borre el archivo original. ¿Puede acceder a los datos a través del enlace simbólico? ¿y a través del enlace duro? ¿Qué ocurre si creamos un archivo con el mismo nombre que el que hemos borrado? ¿Puede acceder a los datos a través del enlace simbólico? ¿y a través del enlace duro?
   e. Intente realizar un enlace duro sobre el archivo de contraseñas del sistema. ¿Qué ocurre? ¿Por qué? Realice un enlace simbólico. ¿Qué ocurre? ¿Por qué?

4. Complete la siguiente tabla:

	¿Es el mismo para todos?	¿Se puede cambiar?
Directorio **/**		
Directorio **.**		
Directorio **~**		

Capítulo 5

Respaldo, Compresión, División y Comparación de Archivos

Respaldo de Archivos

Siempre es recomendable hacer respaldo (backups), de todo, aunque no sea lo más divertido. La pérdida de información es una de las peores catástrofes en el mundo informático, no solo la pérdida de datos en una Base de Datos, sino también podemos perder código/scripts/diseños etc, que nos demandará tiempo volver a escribir.

Hacer un respaldo (backup) de los archivos significa hacer una copia de seguridad de esos archivos para que puedan ser recuperados posteriormente, si fuera necesario. A veces, nos encontramos con la necesidad de hacer una copia de seguridad a un equipo Linux, ya sea desktop o server, podemos respaldar directorios importantes, o hacer un respaldo completo (full backup) de todo el sistema. Los archivos pueden ser copiados en el disco duro, pero la mayoría de los casos se hacen los respaldos en CD, pendrive u otros dispositivos de respaldo, así los datos están protegidos si el disco duro se daña físicamente.

Definiremos tres tipos de copias de seguridad:

- Copias de seguridad completas.
- Copias de seguridad incrementales.
- Copias de seguridad diferenciales

Las **copias de seguridad completas** copian toda una unidad de disco duro, o todos los archivos que se le indiquen. Es la primera copia que debe realizarse *full backup,* la que luego tomarán como base el método de backup incremental o diferencial.

Las **copias de seguridad incrementales** copian sólo aquellos archivos que han cambiado desde la última copia completa o incremental, reduciendo la cantidad de información a copiar en cada proceso. En el caso anterior de tener una copia completa el domingo, el lunes se copiarán las novedades respecto al domingo, y el martes las novedades respecto a la copia del lunes, con la consiguiente reducción de tamaño de copia.

Las **copias de seguridad diferenciales** sólo copian los archivos que han sido creados o modificados desde la última copia completa. Esto quiere decir que si hicimos una copia completa el domingo, el lunes, el martes y los días siguientes, se copiarán aquellos archivos modificados o creados desde el domingo (ultima copia full backups). Las copias diferenciales se van haciendo más grandes ya que reflejan los cambios desde una marca fija en el tiempo.

Ante la necesidad de recuperar los datos, debemos tener en cuenta lo siguiente:

- si vamos a **recuperar archivos de copias incrementales**, necesitaremos la copia completa base y todas las copias incrementales desde la misma hasta la fecha de restauración. Si partimos de incrementales necesitaríamos la copia completa y las incrementales a partir de la misma.

- si tratamos de **recuperar archivos desde una copia diferencial**, necesitaremos la última copia completa de referencia para las copias diferenciales, y la copia diferencial de la fecha que queremos recuperar.

La forma más fácil de hacer un respaldo de forma manual, es con la herramienta **tar** que está disponible en casi cualquier distribución y que nos crea un archivo que opcionalmente puede ser comprimido y es fácil de usar.

tar	genera un archivo .tar de **respaldo o** backup de uno o más archivos.

La palabra **tar** viene de la utilidad **t**ape **ar**chive (archivo de cinta) ya que las copias de seguridad en los sistemas Unix se hacían en una unidad de cinta, pero obviamente en la actualidad

también se pueden copiar en otro dispositivo. El comando *tar* (empaquetar) es muy utilizado normalmente para respaldar archivos, y está disponible en todos los sistemas Linux, lo que permite que sus copias puedan ser utilizadas en cualquier sistema si es necesario. Este comando genera un archivo empaquetado, el que por convención se denominara con la extensión .tar, para que cualquier usuario que vea un archivo con esta extensión reconozca que se ha utilizado el comando *tar* para guardar uno o un grupo de archivos como un único archivo paquete.

Sintaxis:

```
tar [-opciones] archivo.tar [archivo1] [archivo2] ...[archivoN]
```

Opciones:

c	le dice a tar que cree un nuevo archivo de respaldo.
v	(verbose mode) muestra el progreso mientras se crea el archivo imprimiendo en pantalla los nombres de los archivos según se archivan.
f	le dice a tar que el siguiente argumento es el nombre del archivo a crear.
p	permite conservar los permisos de los archivos
z	realiza la compresión de los archivos (ver más adelante)
d	busca diferencias entre los archivos
r	agrega los archivos al final de un archivo
t	lista el contenido del archivo de backups
x	extrae archivos desde un archivo de backups.
X FILE	excluye del backup los archivos contenidos en una lista almacenada en FILE
--exclude='archivo'	excluye del backup el archivo indicado.
-g	crea backup incremental es decir respecto al backup anterior
-N	crea backup diferencial es decir respecto a un full backup.

El resto de argumentos son las rutas con los nombres de archivos y/o directorios a empaquetar. Como siempre podemos encontrar más opciones en *man tar*.

La primera opción de *tar* debe ser la letra **c**, **x**, ó **t** ya que **c** implica crear, **x** extraer un archivo ya creado y **t** mostrar el contenido del archivo tipo tar.

Ejemplo 1

```
$ tar -cvf  copia.tar /home/usuario1
```

Crea un respaldo de los archivos del subdirectorio perteneciente al *usuario1* en un archivo llamado *copia.tar* y se visualiza el proceso de empaquetado

Entonces el comando `$ tar -xvf copia.tar`

Extrae el archivo *copia.tar* en el directorio actual. Esto puede ser peligroso ya que cuando se extraen archivos de un archivo tar, los archivos antiguos se sobrescriben. Por lo que es importante conocer dónde se deben desempaquetar los archivos.

Para ello el comando *tar* tiene la opción ***t***

```
$ tar -tvf copia.tar
```

Se muestra por pantalla, un "índice" del archivo tar antes de desempaquetarlo. De esta forma se puede ver qué directorio se utilizó como origen de los nombres de los archivos, y se puede extraer el archivo desde la localización correcta.

Ejemplo 2

Realizar una copia de respaldo del directorio *respaldo* en el cual tenemos dos archivos *lista_com* y *lista_usu*.

```
$ tar -cvf copiaguion.tar respaldo
```

Obtenemos la salida:

```
alumno@wheezy:~$ tar -cvf copiaguion.tar respaldo
respaldo/
respaldo/lista_usu
respaldo/lista_com
alumno@wheezy:~$ ls copiaguion.tar
copiaguion.tar
```

Supondremos luego que borramos accidentalmente el directorio *respaldo* por lo que tendremos que recuperarlo de nuestro backup. Por lo tanto la orden será:

```
$ tar -cvf copiaguion.tar respaldo
```

La salida sería:

```
alumno@wheezy:~$ tar -xvf copiaguion.tar
alumno@wheezy:~$ tar -cvf copiaguion.tat respaldo
respaldo/
respaldo/lista_usu
respaldo/lista_com
alumno@wheezy:~$ ls respaldo
lista_usu    lista_com
```

Habremos recuperado el directorio *respaldo* y su contenido.

** Los archivos más grandes de 2GB no están soportados por ISO9660 y puede que no sean restaurables. Por lo que se sugiere no grabar simplemente un DVD con un archivo .iso enorme, sino que se debería dividir usando el comando split descripto en páginas posteriores.

El comando *rsync* es una herramienta que permite hacer copias de seguridad, es rápida y versátil. Permite hacer copias localmente, desde o hacia otro host sobre cualquier shell remoto. Utiliza el algoritmo de transferencia Delta, que reduce la cantidad de datos enviados.

Compresión de archivos

Comprimir archivos significa codificar sus contenidos para que sean de un tamaño menor y que se puedan restaurar a su tamaño y contenidos originales cuando se desee. El tamaño del archivo resultante luego de la compresión dependerá de los datos que haya dentro del mismo. Por ejemplo los archivos de texto se pueden comprimir, en relación a un archivo binario, a un tamaño menor que éste ya que los textos contienen muchos caracteres repetidos y los binarios son datos aleatorios.

Existen dos motivos fundamentales por lo que resulta útil comprimir archivos:

- Ocupan menos espacio en disco, podemos comprimir archivos que se usan raramente pero que no deseamos descartarlos.
- Los archivos comprimidos pueden transferirse a otro sistema en un tiempo menor.

En Linux tenemos comandos disponibles para realizar la compresión y descompresión de archivos. Entre ellos encontramos ***gzip*** y el comando ***tar*** con su opción ***z.***

La opción z del comando tar

Esta opción empaqueta y comprime un archivo en una sola acción, lo que puede hacer las cosas más rápidas y cómodas, por lo que es muy usada. Para ello el comando ***tar*** solo hace el empaquetado y es el comando ***gzip*** el que realiza la compresión. En este caso nosotros no tenemos que llamar a ***gzip***, sino que ya lo hace directa e internamente el comando ***tar***.

Sintaxis:

```
tar -czvf [archivo.tgz] [archivo origen]
```

El nombre del archivo comprimido tiene extensión **.tgz**, que indica que estará empaquetado y comprimido.

Ejemplo

```
$tar -czvf copiacomprimida.tgz /home/usu2/textos
```

Se crea el archivo *copiacomprimida.tgz* con el respaldo de los archivos del subdirectorio */home/usu2/textos* y se visualiza el proceso de empaquetado.

Descomprimir y desempaquetar con *tar*

Si por el contrario queremos restaurar un backup, es decir, descomprimir y desempaquetar en un solo paso, tenemos que hacerlo con la opción z del comando tar.

Sintaxis:

```
tar -xzvf archivo.tgz  directoriodestino
```

Ejemplo

```
$ tar -zxvpf fullbackup.tgz  ./
```

Que extrae los contenidos del archivo fullbackup en el directorio actual, descomprimiendo al mismo tiempo y conservando los permisos (opción -p).

Comprimir y Descomprimir con gzip y gunzip

gzip comando que comprime archivos.

Este comando suele efectuar una mejor compresión que el comando *compress* y agrega el sufijo ***.gz*** al nombre del archivo comprimido.

Sintaxis:

```
gzip [-opciones] archivo
```

Opciones:

−c --stdout --to-stdout Escribe la salida en la salida estándar; deja los archivos originales tal cual. Si hay varios archivos de entrada, la salida consiste en una secuencia de miembros comprimidos independientemente.

−d --decompress –uncompress Descomprime.

−f –force Fuerza la compresión o descompresión incluso si el archivo tiene varios enlaces o si el archivo correspondiente ya existe (sin esta opción el sistema pide verificación antes de sobrescribir), o si los datos comprimidos se leen de, o se escriben en, una terminal. Si los datos de entrada no están en un formato reconocido por ***gzip,*** y si se ha dado también la opción --stdout, copia los datos de entrada a la salida estándar sin cambiarlos.

−l –list Para cada archivocomprimido, lista los siguientes campos:
compressed: el tamaño del archivocomprimido
uncompr: el tamaño del archivodescomprimido
ratio: relación de compresión (0.0% si no se conoce)
uncompressed_name: nombre del archivodescomprimido

--verbose, se muestran asimismo los siguientes campos:
method: método de compresión
crc: el CRC de 32 bits de los datos descomprimidos
date & time: tiempos del archivodescomprimido

−r –recursive Recorre la estructura de directorios recursivamente. Si cualquiera de los nombres de archivoespecificados en la línea de órdenes son directorios, ***gzip*** descenderá en el directorio y comprimirá todos los archivosque encuentre allí (o los descomprimirá en el caso de ***gunzip***).

−# --fast –best Regula la velocidad de la compresión según el dígito especificado **#**, donde **−1** ó **−−fast** cuando se desea mayor velocidad de compresión aunque el archivo que se obtiene, no sea tan pequeño y **−9** ó **−−best** el objetivo de *gzip* es obtener el archivo más pequeño posible, lo que le llevará más tiempo. El nivel de compresión predeterminado es **−6**.

Ejemplo

```
$ gzip -9 copia.tar
```

comprimirá *copia.tar* y lo dejará como *copia.tar.gz*, que es la versión comprimida del archivo. La opción -9 le dice a gzip que utilice el mayor factor de compresión.

zcat Muestra contenido de un archivo comprimido

Sintaxis:

```
zcat nom-archivo.gz
```

Es posible mostrar el contenido de un archivo que está comprimido sin tener que descomprimirlo, esto se realiza con el comando ***zcat.***

Ejemplo

```
$ gzip lista1
```

se obtiene el archivo comprimido ***lista1.gz****,* entonces para ver el contenido de este sin descomprimirlo

```
$ zcat  texto1.gz
```

gunzip comando que se utiliza para descomprimir un archivo comprimido por *gzip* o la opción zdel comando tar.

Es equivalente a usar *gzip* con la opción *-d*. El archivo comprimido es eliminado, y se deja el original sin la extensión. Las opciones de *gunzip* son las mismas que las de *gzip*, menos la opción *-d* que va implícita (*-c, -l, -r, -t, -v, -S*).

Sintaxis:

```
gunzip [-opciones] archivo.gz ó archivo.tgz
```

Ejemplo

```
$ gunzip copia.tgz
$ ls
    copia.tar
```

En este caso el archivo copia fue empaquetado y comprimido en un solo paso (copia.tgz) y luego el archivo es solo descomprimido pero no desempaquetado.

Dividir un archivo en múltiples archivos más pequeños

A veces resulta muy útil dividir archivos de gran tamaño en varios de tamaño menor por ejemplo cuando se quiere guardarlo en diferentes discos o cuando se desea enviar archivos por mail o en general a través de la red, es mucho más sencillo y práctico enviar archivos pequeños y transferirlos en forma separada, luego con el comando *cat* es fácil reconstruir el original.

split comando que divide un archivo en un grupo de archivos que contenga cada uno de ellos la misma cantidad de líneas o longitud. El número de líneas por defecto es de 1000.

Sintaxis:

```
split [-opción] archivo
```

Opciones:

- ***-a N*** cantidad de caracteres que tiene el sufijo al generar los nombres de los archivos.
- ***-b SIZE*** genera archivos de tamaño indicado en SIZE. Esta opción permite el corte de las líneas si fuera necesario. El tamaño (SIZE) es un número entero y se especifica con los siguientes sufijos: b 512, kB 1000, K 1024, MB 1000*1000, M 1024*1024, GB 1000*1000*1000, G 1024*1024*1024. Por ejemplo 10k es 10*1024. Las unidades K,M,G,T,P,E,Z,Y (multiplican por 1024) o KB,MB,….. multiplican por 1000el número entero.
- ***-C SIZE*** escribe un máximo, por SIZE (tamaño), en bytes sin cortar las líneas en el archivo de salida.
- ***-d*** indica que los sufijos son numéricos. Los archivos generados fueron: x00, x01, x02, x03, x04, x05, x06, x07, x08 y x09.
- ***-l NUMBER*** Con esta opción decidimos cuántas líneas tiene cada archivo de salida.

Ejemplo 1

Si tenemos el archivo *lis.txt* de 19MB, podemos dividirlo en archivos de 2MB es decir 2x1024x1024 y guardar los archivos generados en el subdirectorio divide. (Puede crear el archivo como *ls -lR /usr > lis.txt* y el subdirectorio *mkdir ./divide*)

```
$split -b 2M lis.txt divide/lis       ó
$split --bytes=2M lis.txt divide/lis
```

La salida será

```
alumno@wheezy:~$ split -b 2M lis.txt divide/lis
alumno@wheezy:~$ ls -lh divide
total 19M
-rw-rw-r--  1 alumno alumno 2,0M mar 8 11:53 lisaa
-rw-rw-r--  1 alumno alumno 2,0M mar 8 11:53 lisab
-rw-rw-r--  1 alumno alumno 2,0M mar 8 11:53 lisac
-rw-rw-r--  1 alumno alumno 2,0M mar 8 11:53 lisad
-rw-rw-r--  1 alumno alumno 2,0M mar 8 11:53 lisae
-rw-rw-r--  1 alumno alumno 2,0M mar 8 11:53 lisaf
-rw-rw-r--  1 alumno alumno 2,0M mar 8 11:53 lisag
-rw-rw-r--  1 alumno alumno 2,0M mar 8 11:53 lisah
-rw-rw-r--  1 alumno alumno 2,0M mar 8 11:53 lisai
-rw-rw-r--  1 alumno alumno 928K mar 8 11:53 lisaj
```

Se crearon 10 archivos con un máximo de 2MB cada uno, el último tiene sólo 928k:

Si necesitamos dividir el archivo en archivos de 2MB pero teniendo el cuidado de no cortar las líneas, el comando es:

```
$split -C 2M lis.txt divide/lis
```

Ejemplo 2

```
$split  informe
```

Se crean los archivos: `xaa  xab  xac  xad`

Los tres primeros archivos *xaa* hasta *xac* tendrán 1000 líneas cada uno, que es la cantidad de líneas que toma por defecto el comando, el último, *xad* posiblemente tendrá menos porque almacena las líneas restantes del archivo.

Ejemplo 3

Si se desea, es posible especificar el número de líneas que hay que poner en cada segmento o parte del archivo, por ejemplo 800 lineas:

```
$split -l 800 informe informe-
```

Creará los archivos: informe-aa informe-ab informe-ac informe-ad

Para volver a crear el archivo original solo es necesario utilizar el siguiente comando:

```
$cat informe-*  > nuevoarchivo
```

Comparar archivos

Linux utiliza simples archivos de texto en una gran variedad de funciones importantes, incluidas las tablas de configuración del sistema, el código fuente del programa C, las páginas HTML de web y los datos del usuario. En Linux los archivos de texto tienen objetivos importantes por ello existen utilidades que permiten comparar estos archivos de texto, como por ejemplo el comando *diff.*

diff	comando que permite ver las diferencias entre dos archivos.

Sintaxis:

```
diff  -opciones archivo1 archivo2
```

El resultado del comando representa los cambios requeridos para hacer que el archivo original (archivo1) se convierta en el nuevo archivo (archivo2).

Si *diff* no muestra ninguna salida, los archivos son idénticos. Cualquiera sea la salida de *diff* significa que los archivos son diferentes. Cuantas más salidas hay, mayor es el número de líneas que son diferentes. Si uno de los argumentos del comando es un guion "**-**", el comando compara el archivo con el texto leído desde la entrada estándar.

Si en los argumentos de la línea de comandos, ***archivo1*** es un directorio y ***archivo2*** es un archivo común, el comando diff compara el archivo en el directorio que tenga el mismo nombre que ***archivo2*** y viceversa.

Si ***archivo1*** y archivo2 son directorios, entonces diff se ejecutará sobre cada archivo que exista en ambos directorios.

Opciones:

- **-*a*** trata a todos los archivos como de texto y los compara línea por línea.
- **-*b*** ignora los espacios y los tabuladores en la línea, cuando se quiere comparar dos archivos de texto que tienen márgenes diferentes.
- **-*B*** no considera cambios consistentes en sólo insertar y borrar líneas en blanco.
- **-r** cuando compara los archivos en los directorios, lo hace en forma recursiva.
- **-*e*** genera un edit script. Un archivo editable.

En la página del manual correspondiente a *diff* puede encontrar más opciones.

Ejemplo 1

```
$diff notas.1  datos.1
3c3
<     ..... texto de linea 3   notas.1
-------
>     ..... texto de linea 3   datos.1.......

 . . . . .y asi sucesivamente . . . . . .
```

La salida de este comando nos dice que el archivo notas.1 difiere del archivo datos.1 en la tercera línea, y para que ambos sean iguales habría que cambiar la línea 3 del primero por la línea 3 del segundo (3c3). Con el simbolo "<" se indica el contenido de la línea del archivo de la izquierda, y con el simbolo" >" se indica el contenido respectivo del archivo de la derecha. Y así para el resto.

Por lo tanto se muestran todas las diferencias entre ambos archivos, y también que debería hacerse sobre el primero de ellos, para que ambos sean iguales.

En este formato de salida tradicional, **a** sustituye a *añadido*, **d** sustituye a borrado (*d*eleted) y **c** sustituye a cambiado. Los números de línea del archivo original aparecen antes de a/d/c y los del archivo modificado después. Los paréntesis angulares aparecen al comienzo de las líneas que son

añadidas, borradas o cambiadas. Las líneas añadidas se incluyen en el archivo original para aparecer en el archivo nuevo. Las líneas borradas se eliminan del archivo original para ser borradas en el archivo nuevo.

Por defecto, las líneas comunes a los dos archivos no se muestran. Las líneas que se han movido se muestran como añadidas en su nuevo lugar y como borradas en su antiguo lugar.

Ejemplo 2

Tenemos dos directorios *divide y divide2* con prácticamente la misma información, pero necesitamos conocer claramente cuáles son esas diferencias si es que las hay, entonces ejecutamos el comando:

```
$ diff divide divide2
```

La salida será:

```
alumno@wheezy:~$ ls –l divide2
total 4108
-rw-r--r--  1 alumno alumno 2096807 mar 8 11:53 lisaa
-rw-r--r--  1 alumno alumno 2097145 mar 8 11:53 lisab
-rw-r--r--  1 alumno alumno     159 mar 8 12:45 xxaa
-rw-r--r--  1 alumno alumno     185 mar 8 12:45 xxab
-rw-r--r--  1 alumno alumno     213 mar 8 12:45 xxac
alumno@wheezy:~$ ls –l divide
total 5928
-rw-r--r--  1 alumno alumno 2097116 mar 8 10:38 lisaa
-rw-r--r--  1 alumno alumno 2097145 mar 8 11:59 lisab
-rw-r--r--  1 alumno alumno 1875951 mar 8 11:59 lisac
alumno@wheezy:~$ diff divide divide2
diff divide/lisaa divide2/lisaa
770,771d769
<lrwxrwxrwx  1  root  root     6  mar  2  2017 rtstat ->lnstat
<-rwxr-xr-x  1  root  root 34408  ene  26 2017 runcon
915,917d912
<-rwxr-xr-x  1  root  root 61396  jun  30 2016 traceroute.db
<-rwxr-xr-x  1  root  root  1186  jun  30 2016 traceroute.nanog
<-rwxr-xr-x  1  root  root 86144  jun  14 2016 transset
Solo en divide: lisac
Solo en divide2: xxaa
Solo en divide2: xxab
Solo en divide2: xxac
```

Al comparar los directorios *divide* y *divide2*, analiza primero el archivo *lisaa* que está en ambos e indica que las líneas 770 a 771 y 915 a 917 están en *divide/lisaa* pero no en *divide2/lisaa* y las muestra (< corresponde a *divide/lisaa*), luego indica que el archivo *lisac* está sólo en el directorio divide y los archivos *xxaa, xxab, xxac* están sólo en el directorio *divide2.*

cmp	compara dos archivos byte por byte.Se utiliza para comparar cualquier tipo de archivos.

Sintaxis:

```
cmp archivo1 archivo2 [salto1] [salto2]
```

Si *cmp* vuelve al indicador de shell sin mostrar ningún mensaje o cero, significa que los archivos son idénticos, devuelve 1 si son diferentes y 2 si hubo algún problema en la comparación. El comando *cmp* sin opciones se detiene cuando encuentra la primera diferencia, por lo que funciona más rápido que *diff.* Si en archivo2 utilizamos guion **-** o nada, lee desde la entrada estándar. Compara los archivos hasta que encuentra en alguno de los dos la marca de final de archivo.

Opcionalmente podemos indicar ***salto1*** *y* ***salto2*** indican el número de bytes a saltar al comienzo de cada archivo. El valor en **salto** (numero entero) puede ser acompañado por las siguientes letras KB, K, MB, M, GB, G y así para T, P, correspondientes a las unidades 1000, 1024, 1.000.000, 1.048.576, 1.00.000.000, 1.073.741.824 respectivamente.

Opciones:

–l muestra todas las diferencias que hay entre los dos archivos

Encontrará más opciones en la página de manual.

Ejemplo:

```
$cmp notas.1  datos.1

notas.1  datos.1 differ: char 23, line 5
```

Indica que ambos archivos tienen su primera diferencia en el carácter 23 de la línea 5.

Actividad 1

1. Cree el archivo *lista_com* que contenga los nombres de los comandos (con sus atributos principales) que pueden ejecutar los usuarios del sistema Linux.

```
ls -l  /bin > lista_com
```

2. Cree el archivo *lista_usu* que contenga los usuarios activos del sistema. Verificar la existencia y el contenido de los archivos creados.

```
who > lista_usu
ls   -l lista_com lista_usu
more lista_com
more lista_usu
```

3. Realice una copia de seguridad de los archivos creados y llámela *copia.tar.* Verifique si los archivos originales todavía existen.

```
tar cvf copia.tar  lista_com lista_usu
 ls -l lista_com lista_usu

# La salida del comando tar muestra los nombres de los archivos según se van
copiando en el respaldo, y los archivos originales permanecen sin alteración.
```

4. Muestre el contenido del archivo de respaldo creado.

```
 tar tvf copia.tar

# La salida de este comando es casi idéntica a realizar un ls -l de los
archivos guardados.
```

5. Extraiga los archivos originales del archivo de respaldo, en el sudirectorio *backup* creado en su directorio de login.

```
mkdir backup
 cd backup
 tar xvf  ../copia.tar

# Recordemos que la opcion x, desempaqueta los archivos respaldados en el
directorio actual por lo que es necesario tener creado el subdirectorio
destino con anticipación. Luego cambiarse a ese directorio y recién allí
extraer los archivos del respaldo. Este comando reemplaza los archivos
existentes con el mismo nombre por lo que es necesario, tener muy en cuenta
dónde extraemos esos archivos.
```

6. Realice una copia de seguridad comprimida de los archivos de su directorio de conexión en un archivo llamado *backup.tgz.* Verifique si los archivos originales todavía existen.

```
cd
tar czvf backup.tgz  ./
ls -l

#Con el comando cd me muevo desde el directorio de trabajo al de conexión
Luego se creará un respaldo de los archivos del subdirectorio actual en un
archivo llamado backup.tgz y se visualizará el proceso de empaquetado.
Al finalizar los archivos originales permanecen sin alteración.
```

7. Muestre el contenido del archivo de respaldo creado.

```
 tar tvf backup.tgz

#La salida de este comando es casi idéntica a realizar un ls -l de los
archivos guardados.
```

8. Extraiga del archivo *backup.tgz* los archivos comprimidos, en el subdirectorio COPIAScreado en su directorio de conexión.

```
mkdir COPIAS
 cd COPIAS
 tar xzvf  ../backup.tgz
```

#Recordemos que la opción x, desempaqueta los archivos respaldados en el directorio actual, por lo que es necesario tener creado el subdirectorio destino con anticipación; luego cambiarse a ese directorio y recién allí extraer los archivos del respaldo. Este comando reemplaza los archivos existentes con el mismo nombre, por eso, se hace necesario tener en cuenta dónde extraemos esos archivos.

9. Comprima solamente el archivo *lista_com* como *lista_com.gz.* Luego verifique si el archivo original todavía existe.

```
gzip lista_com
ls -l
```

 # Ud. sabe que al comprimir un archivo con el comando gzip automáticamente se agrega la extensión .gz, por lo que gzip es el comando a usar. Observe también que el archivo compactado reemplaza al original.

10. Muestre el contenido del archivo comprimido *lista_com.gz* y luego descomprímalo.

```
 zcat lista_com.gz
 gunzip lista_com.gz
```

 # No es necesario descomprimir un archivo para ver el contenido del mismo, gracias al comando zcat es posible esto.
gzip tiene su comando inverso gunzip y al ejecutarlo sobre el archivo compactado lo descomprime y es reemplazado por el archivo original.

11. Divida el archivo lista_com en archivos de 15 líneas cada uno. Verifique los archivos existentes.

```
split –15 lista_com
ls –l
# Se han creado los archivos xaa – xab – xac, etc pero el archivo
 original lista_com sigue existiendo.
```

12. Cree el archivo *coman_dos* que contenga los nombres de los comandos que pueden ejecutar los usuarios del sistema Linux y luego agregue al mismo archivo el número de i-nodo del archivo creado.

```
ls /bin  > coman_dos
ls –i coman_dos >> coman_dos
```

13. Realice la misma operación pero creando un archivo llamado *coman_tres*

```
ls /bin  > coman_tres
ls –i coman_tres >> coman_tres
```

14. Verifique si hay diferencias entre el archivo *coman_dos* y el archivo *coman_tres*

```
diff coman_dos  coman_tres
```

 # El objetivo de los pasos anteriores era crear dos archivos con una línea de diferencia para realizar una comparación entre ellos. El comando diff muestra el nro. de línea donde está la diferencia y el texto de la línea propiamente dicha.

15. Compare los archivos anteriores y describa la salida obtenida.

```
cmp coman_dos coman_tres

# Como puede observar la salida de este comando muestra la línea y el carácter a partir del cual comienza la diferencia.
```

16. Copie el archivo *coman_tres* al archivo *copia3, v*erifique que la copia exista y luego compárelos. ¿Qué salida obtiene?

```
cp coman_tres copia3
ls  ~/
cmp comandos copia3

# No obtenemos ninguna salida ya que los archivos son idénticos.
```

Actividad 2

1. Luego de iniciar su sesión. Cree el archivo *dispositivos* que contenga los nombres de los dispositivos que existan en el subdirectorio /dev, con sus atributos principales.

```
Login : alumno
Password: *******
ls –l  /dev > dispositivos
```

2. Dividir el archivo *dispositivos* en archivos que comiencen con el prefijo "*device"* de 150 líneas cada uno. Verificar los archivos creados.

```
split –150 dispositivos device-
ls –l device*
```

3. Realice una copia de seguridad de los archivos creados (dispositivos y device*) en *backup2.tar.* Verifique si los archivos originales todavía existen.

```
tar cvf backup2.tar  dispositivos device*
ls –l dispositivos device*
```

4. Muestre el contenido del archivo de respaldo creado.

```
tar tvf backup2.tar
```

5. Comprima el archivo *backup2.tar* como un archivo ***.gz*** con la mayor velocidad de compresión. Verifique el archivo creado.

```
gzip –1 backup2.tar

#Como observó utilizamos el comando gzip con la opción –1 que permite mayor velocidad de compresión aunque el archivo que se obtiene no sea tan pequeño. El archivo obtenido conserva la extensión .tar y agrega .gz
```

6. Descomprima el archivo creado anteriormente.

```
gunzip  backup2.tar.gz

#Este comando vuelve el archivo compactado al archivo que le dio origen desapareciendo la extensión .gz
```

7. Extraiga los archivos originales del archivo de respaldo, en el subdirectorio *recupero* de su directorio de login.

```
mkdir recupero
cd recupero
tar xvf  ../backup2.tar

#Recuerde tener cuidado de no pisar los archivos existentes cuando extraiga los archivos, por eso generalmente se utiliza un subdirectorio nuevo.
```

Actividad 3

1. Cree un archivo de respaldo de los archivos del directorio /etc

2. Comprima los archivos de /bin que comiencen con la letra **m**

3. Realice una copia de seguridad de los archivos creados en *seguridad.tar.* Muestre el contenido del archivo de respaldo creado.

4. Divida el archivo de respaldo creado en archivos de 20 líneas cada uno. Verifique los archivos existentes.

Actividad 4

Situación problemática

Un administrador de sistemas de la Obra Social XEDR recibe el pedido del área de Contabilidad, solicitando la instalación de algunas aplicaciones nuevas que involucran el uso de cierta cantidad de espacio físico en el file system.

Solución planteada

Como primera medida del administrador de sistemas debe:

- *Conocer cuánto espacio ocuparán en el disco las nuevas aplicaciones. Suponemos que ocupan 120MB.*
- *Realizar un control de todos los sistemas de archivos (file system) en uso, para asegurarse que no haya problemas de espacio a la hora de instalar una nueva aplicación. La falta de espacio puede provocar errores en las aplicaciones o errores de sistema e incluso corrupción de archivos. Para realizar esto se utiliza herramientas tales como df y du (capitulo 10), que identifica a los archivos que más espacio ocupan.*
- *Hacer una copia de respaldo comprimida de los directorios antiguos y de poco uso, para luego eliminarlos de su ubicación original y poder hacer uso del espacio generado. Decide comprimir las bases de datos de facturación de los año 2104 y 2015 (/xedr/facturacion2014 y /xedr/facturacion2015). Ya que no se consultan.*

Luego de realizar estas tareas el administrador dispone autorizar el pedido de instalación de las nuevas aplicaciones.

Nota: Generar respaldos, comprimir y descomprimir archivos es una de las tareas más comunes que vamos a tener que realizar en Linux por línea de comandos. Podemos hacerlo mediante diversas herramientas, pero dos van a estar disponibles en cualquier distribución: tar y gzip que se usan de manera complementaria.

tar -- empaqueta varios archivos, en un único archivo,
gzip -- hace la compresión.
La opción z del comando ***tar*** *empaqueta y comprime en una sola acción.*

De todos modos, tar simplemente hace el empaquetado y es gzip el que realiza la compresión. Simplemente que nosotros no tenemos que llamar a gzip, sino que ya lo hace directa e internamente el comando tar.

#tar czvf F14y15.tgz /xedr/facturacion2014 /xedr/facturacion2015

El nombre del archivo comprimido tiene extensión .tgz, que indica que estará empaquetado y comprimido. Esta acción libera suficiente espacio en el disco como para realizar la instalación.
Si por el contrario, queremos descomprimir y desempaquetar en un solo paso, tenemos que hacerlo con la opción z del comando tar.

Capítulo 6

Filtros

Filtros

Bajo este nombre podemos ubicar a una familia de comandos: *grep, sort, find, locate, wc*, *cut sed*, *tr, uniq, tee*. Permiten buscar y mostrar cadenas y archivos, ordenar archivos, mostrar parte de un archivo, etc. Es decir, muestran una salida modificada, sin cambiar los datos de entrada.

grep busca en uno o más archivos una **cadena de caracteres** o patrón y muestra las líneas de texto que lo contienen.

Es uno de los programas más potentes del sistema operativo Linux. Existen variaciones del mismo comando: *egrep, fgrep (grep –F), pgrep, rgrep (grep – r).*

El comando *grep* es de gran utilidad para encontrar ocurrencias de variables en programas y para seleccionar partes de la salida de un programa. Utiliza expresiones regulares para definir la consulta. Puede manejar archivos, directorios (y subdirectorios), o la entrada estándar (stdin). Por omisión, grep, muestra todas las líneas de los archivos en dónde figura la cadena de caracteres que se desea buscar.

Sintaxis:

```
grep [opciones] patrón [archivo1 archivo2 …]
```

Opciones:

-n muestra la línea en donde encontró el texto y el número de línea dentro del archivo.
-v invierte la búsqueda, o sea muestra las líneas que no contienen el patrón.
-i ignora la diferencia entre mayúsculas y minúsculas en las búsquedas.
-c muestra solamente el número de líneas coincidentes
-l si se busca en un grupo de archivos, lista los nombres de los archivos donde se encuentran coincidencias sin mostrar el texto.
-L muestra sólo el nombre de los archivos que *no* contienen la cadena o patrón.
-r permite hacer búsquedas de forma recursiva dentro de los directorios que se encuentran en la ruta de búsqueda.

Si el archivo en el que desea realizar la búsqueda se encuentra en otro directorio, debe especificar la ruta (absoluta o relativa) de acceso al mismo. También puede usar metacaracteres para buscar en múltiples archivos.

Ejemplo 1

```
$grep  fragmentación  memoria

en la asignación de memoria a los procesos es la fragmentación.
a la fragmentación  externa como el espacio de memoria que no puede ser
            :           :
```

Esta línea de comando genera un listado de las líneas del archivo *memoria* que contienen la palabra “fragmentación”.

Ejemplo 2

```
$grep frag *.txt

      mem.txt: la desventaja principal es la fragmentación.
      Disc.txt: el disco esta fragmentado…
         :
```

En este ejemplo grep busca la cadena “frag” en todos los archivos con extensión *txt*, y muestra el nombre del archivo en donde encontró la cadena y la línea que la contiene.

Expresiones Regulares

El comando *grep* no sólo puede manejar cadena de caracteres como patrones de búsqueda, sino también cadenas denominadas *expresiones regulares*.

Las *expresiones regulares* se especifican dando un significado especial a ciertos metacaracteres utilizados por el shell. Es decir que un metacaracter utilizado por el shell tiene distinta utilidad si es manejado por el comando grep.

Los patrones de búsqueda que utilizan expresiones regulares, van entre comillas simples para evitar que el shell los interprete como metacaracteres. Los patrones de búsqueda que tienen más de una cadena de caracteres pueden ir entre comillas simples o dobles.

Algunos de sus parámetros para construir las expresiones regulares son:

^	Indica inicio de línea
$	Indica final de línea
****	Evita dar significado especial a un carácter por ej. Para ^ haríamos: \^
[ ]	Indica un conjunto de caracteres o un rango de letras o números
[^]	Indica la negación de un conjunto de caracteres
. (punto)	Indica cualquier carácter
***** (asterisco)	Indica que puede existir el carácter o expresión anterior ninguna o varias veces
\{x,z\}	Indica que el carácter o expresión anterior puede existir entre x y z veces
\{x\}	Indica que el carácter o expresión debe repetirse exactamente x veces
\{x,\}	Indica que el carácter o expresión debe repetirse x veces o más

Los metacaracteres **^** y **$** buscan al inicio o al final de la línea, respectivamente. Reciben el nombre de anclas, pues anclan un patrón a un determinado lugar de la línea.

Veamos cómo trabajan algunas expresiones regulares. Para ello, vamos a crear primero un archivo para utilizar en los ejemplos:

Ejemplo 1

```
$ls -l  /home/garcia  >  directorios

$grep '^d' directorios
   drwxrw--r--    3   rgarcia     rgarcia    ...  practicos
   drwxr--r-x     2   rgarcia     rgarcia    ...  monografias
```

El comando anterior hace un listado de todas las líneas del archivo *directorios* que tienen como primer carácter una *d.* Si recordamos, en el archivo *directorios* tenemos un listado extendido de los archivos y directorios de */home/garcia*, por lo que en definitiva, con el comando *grep* generamos un listado de los subdirectorios del directorio *garcia*.

Ejemplo 2

Supongamos ahora que en el directorio */home/garcía* tenemos varios archivos denominados *padronaa, padronab*, etc, que habíamos generado con el comando *split,* y necesitamos un listado extendido de ellos. Entonces, una de las opciones a utilizar puede ser la siguiente:

```
$grep 'padron..$' directorios
-rw-r--r--  1  rgarcia  rgarcia  2048 May  10 2003   padronaa
-rw-r--r--  1  rgarcia  rgarcia  2048 May  10 2003   padronab
-rw-r--r--  1  rgarcia  rgarcia  2048 May  10 2003   padronac
-rw-r--r--  1  rgarcia  rgarcia  2048 May  10 2003   padronad
```

El símbolo **$** ancla la búsqueda al final de cada línea del archivo *directorios*. Es decir que muestra las líneas en cuyo último campo esté la cadena *padrón*. Los puntos, corresponden a una expresión regular y reemplaza cada uno de ellos a cualquier carácter que esté en esa posición de la cadena a buscar.

El punto **"."** reemplaza a cualquier carácter, tiene la misma utilidad que el metacarácter **?** del shell.

Ejemplo 3

```
$grep '^...x' directorios
drwxrw--r--  3  rgarcia  rgarcia    ...     practicos
-rwxr--r--   1  rgarcia  rgarcia    ...     buscar
drwxr--r-x   2  rgarcia  rgarcia    ...     monografias
-r-xr--r--   1  rgarcia  rgarcia    ...     ver
             :                          :
```

El símbolo **^** ancla la búsqueda al comienzo de cada línea, y los tres puntos corresponden a cualquier combinación de los tres primeros caracteres de la línea. Es decir que lista los archivos y directorios en los cuales el dueño tiene permiso de ejecución.

El punto se puede utilizar al comienzo, medio o final de cualquier cadena o patrón de búsqueda.

Ejemplo 4

Supongamos ahora que necesita buscar los datos de un deudor en particular, el Sr. Simbber, pero no recuerda cómo se escribe, sólo recuerda que comienza con "Sim", entonces puede utilizar el asterisco en combinación con el punto de la siguiente manera:

```
$ grep 'Sim.*' deudores
 Jorge Sim          sistemas     36.00
 Tomas Simberman    sistemas     37.00
Esteban Simbber     contabilidad 48.00
              :                       :
```

El comando muestra todas líneas en donde encontró cualquier secuencia de caracteres que contenga "Sim". Debemos tener en cuenta que el asterisco sólo se aplica a un carácter (sólo afecta al carácter que está a su izquierda). Luego, cualquier número de repeticiones de un carácter incluye al cero, por lo que es posible que el comando anterior reconozca la cadena "Sim" como un apellido válido, sin que haya algún carácter después de estos tres.

El comando grep maneja los rangos de forma muy parecida al shell, de modo que [a-g] se acopla con cualquier minúscula de la letra **a** a la **g**.

Ejemplo 5

```
$grep 'dato[0-9]' directorios
drwxrw--r--  3   rgarcia       rgarcia    ...      dato1
-rwxr--r--   1   rgarcia       rgarcia    ...      dato3
drwxr--r-x   2   rgarcia       rgarcia    ...      dato6
-r-xr--r--   1   rgarcia       rgarcia    ...      dato8
           :                              :
```

Muestra un listado de los archivos de nombre dato y el último carácter es un número del cero al nueve. Si un rango de caracteres comienza con el símbolo ^, el patrón se acopla con cualquier carácter excepto los del rango. Es decir, [^ 0-9] concuerda con cualquier carácter que no sea un dígito.

Ejemplo 6

```
$ls |grep 'a\{1,\}$'
aa
archaa
linaa
linba
linca
linda
línea
```

Filtra de la salida del comando *ls* los nombres que terminan con una o más letras**a.**

Ejemplo 7

```
$ls |grep 'a\{2\}$'
aa
archaa
linaa
```

Filtra de la salida del comando *ls* los nombres de archivos que terminan con exactamente dos letras **a**.

Ejemplo 8

```
$ls |grep 'a\{2,3\}'
  aa
  archaa
  device-aa.gz
  domaa.tar
  linaa
```

Filtra de la salida del comando *ls* los nombres de archivos que tienen 2 o 3 letras**a** seguidas.

egrep busca en uno o más archivos una cadena de caracteres o patrón y muestra las líneas de texto que lo contienen.

Sintaxis:

```
egrep [opciones] patrón [archivo1 archivo2 …]
```

Busca dentro de los archivos un patrón especificado y muestra las líneas que contienen a ese patrón. El patrón se interpreta como una expresión regular. Estas son las mismas que las del comando *grep* y algunas más:

+ Una o más repeticiones del carácter precedente.
? Cero o más repeticiones del carácter precedente.
| Identifica cualquiera de dos o más elementos.
( ) Trata el texto entre paréntesis como un grupo.

Los paréntesis pueden utilizarse para agrupar un conjunto de caracteres que van a ser afectados por una expresión regular, como por ejemplo el *.

Ejemplo 1

```
$egrep '(Sim)+' deudores
Jorge   Sim       sistemas    36.00
Tomas Simberman   sistemas    37.00
Esteban Simbber   contabilidad     48.00
Ignacio SimSim    contabilidad     67.00
        :                  :
```

El **signo +** permite buscar cualquier repetición de los caracteres *Sim*. En este ejemplo el comando e*grep* lista todas las líneas del archivo *deudores* en donde encontró la cadena o cualquier repetición de la misma.

La barra vertical | es un operador OR: *cadena1* | *cadena2*. El comando se acopla con la cadena1 o con la cadena2.

Ejemplo 2

```
$egrep  'dat(osln|enla)' directorios
drwxrw--r--  3   rgarcia rgarcia    ...       datosln
-rwxr--r--   1   rgarcia rgarcia    ...       datenla
```

El comando egrep funciona mucho más rápido que grep cuando se utilizan expresiones regulares, y sobre todo cuando se inspeccionan archivos grandes.

tr reemplaza o elimina un conjunto de caracteres de la entrada estándar (stdin)

Este comando trabaja byte a byte en las operaciones que realiza (eliminación/reemplazo), cambia cada uno de los caracteres especificados en el conjunto inicial por los caracteres especificados en el conjunto final. Trabaja leyendo la información de la entrada estándar y volcando el resultado directamente a la salida estándar. El archivode origen o archivodestino lo especificamos con los caracteres de redirección: **<** ó **>**.

Sintaxis:

```
tr [-opciones]  CONJUNTO1  CONJUNTO2
```

Opciones:

-c, -C, --complement

Usa el complemento del CONJUNTO1. Esto significa que define al CONJUNTO1 como todos los caracteres que no se encuentran en la definición dada por el usuario. Este parámetro es útil para indicar caracteres que no queremos que se vean afectados.

-d, --delete

Borra los caracteres definidos en CONJUNTO1.

-s, --squeeze-repeats

Elimina la secuencia continua de caracteres repetidos, definidos en el CONJUNTO1. Es decir, sustituye un conjunto de caracteres repetidos por uno sólo.

-t, --truncate-conjunto1

Trunca la longitud del CONJUNTO1 según la longitud del CONJUNTO2. Esto hace que si el primer conjunto es más largo que el segundo, solo sean tenidos en cuenta *n* primeros caracteres, siendo *n* la longitud de CONJUNTO2.

Podemos utilizar en CONJUNTOS algunos caracteres y conjuntos predefinidos por el comando **tr**:

\f : Salto de Página.

\n : Nueva línea.

[:lower:] : Letras minúsculas.

[:upper:] : Letras mayúsculas.

Ejemplo 1

```
$tr -d "\n" < deudores
```

Elimina el salto de línea en el archivo *deudores*.

Ejemplo 2

```
$tr -s ':'  ' ' < /etc/passwd > cuentas
```

Reemplaza en cada línea del archivo*/etc/passwd* el carácter **:** por el espacio. La salida del comando se redirecciona al archivo *cuentas*.

Ejemplo 3

```
$ls -l | tr -s " "
```

La opción –s sustituye un conjunto de espacios por uno solo. El comando trabaja sobre la salida del comando ls –l.

Ejemplo 4

```
$tr ' ' '\n' < deudores
```

Reemplaza cada espacio en las líneas del archivo *deudores*, por un salto de línea, los datos se mostrarán en una columna.

Ejemplo 5

```
$tr '\n' ' ' < deudores
```

Reemplaza los caracteres de nueva línea en espacios.

Ejemplo 6

```
$tr -c '[a-z][A-Z][0-9]' ? /etc/group
```

En el primer argumento se indican tres rangos, todas las letras del alfabeto en minúsculas, todas las letras del alfabeto en mayúscula y los 10 símbolos del sistema decimal. Con la opción c el comando tr, reemplaza los caracteres que no se encuentren especificados en los rangos indicados en el primer argumento por un signo de pregunta. Es decir, todo lo que no sean letras o números se reeplaza por **?.**

Ejemplo 7

```
$echo "proceeesos dormidos" | tr -s 'e'
```

Elimina todas las repeticiones de la letra **e** dejando una únicamente ("procesos")

Ejemplo 8

```
$echo "casa" | tr -dc "a"
```

Elimina los caracteres distintos de **a**. Devuelve únicamente las letras 'a'.

Ejemplo 9

```
$echo "procesos bloqueados" | tr -dc "o" | wc -c
```

Devuelve el número de letras **'o'** que contiene el texto

Ejemplo 10

```
$echo "ProcesoS" | tr [:lower:] [:upper:]
```

Convierte el texto que está en minúsculas a mayúsculas.

cut	muestra segmento o porciones de las líneas de texto de un archivo o desde la entrada estándar stdin, para ello utiliza delimitadores para determinar dónde dividir los campos.

Sintaxis:

```
cut -opciones [archivo]
```

Cuando no se incluye el nombre de archivo o se indica -, cut lee de la entrada estándar.

Opciones:

-f --fields=*rango/s*.	Muestra los campos (columnas) indicadas en el o los rangos, que se encuentran delimitadas por el carácter Tabulador.
-c--characters=*rango/s*.	Muestra los caracteres indicados en el o los rango
-b--bytes=*rango/s*.	Muestra los bytes indicados en el o los rangos.
-d--delimiter=*caracter_delimitador*	Especifica el delimitador de campos (entre comillas)
-s	Indica que las líneas que no posean el delimitador (o separador) no sean mostradas.

Para nuestros ejemplos, suponemos creado el archivo ***lista*** con las siguientes líneas (apellido, legajo, documento, edad):

```
Gutierrez   45378       30569025    20
Torres      42963       32987026    18
Albama      40258       32025147    19
Kirmat      45147       33589632    16
```

Ejemplo 1

```
$cut -d' ' -f1 lista
      Gutierrez
      Torres
      Albama
      Kirmat
```

Muestra el primer campo, con la opción –d indicamos que el delimitador es un espacio.

Ejemplo 2

```
$cut -d' ' -f1,3 lista
      Gutierrez   30569025
      Torres      32987026
      Albama      32025147
      Kirmat      33589632
```

Muestra la primera y tercera columna del archivo lista.

Ejemplo 3

```
$cut -d' ' -f2-4 lista
      45378       30569025    20
      42963       32987026    18
      40258       32025147    19
      45147       33589632    16
```

Muestra las columnas 2, 3 y 4 del archivo lista.

Ejemplo 4

```
$cut -c1-13 lista
      Gutierrez 453
      Torres 42963
      Albama 40258
      Kirmat 45147
```

Muestra los primeros 13 caracteres de cada línea del archivo lista.

find permite buscar archivos en el sistema de archivos.

Este comando se utiliza para buscar archivos en el file system, desde un directorio específico y en forma recursiva sobre las ramas inferiores a éste. La búsqueda se realiza de acuerdo al nombre del archivo o a algún atributo de éste.

Sintaxis:

```
find  pathname -opciones argumentos [acciones]
```

Donde *pathname* hace referencia a la ruta absoluta o relativa al directorio donde va a comenzar la búsqueda. Puede indicar más de un directorio inicial.

Por otra parte, –*opciones* hace referencia a los tipos de búsqueda, por ejemplo: buscar por nombre, por permisos, por dueño del archivo, etc. Es decir que las opciones ya no son un carácter sino una palabra.

Opciones:

-name {patrón} buscamos los archivos que coincidan con el patrón.
-iname ignora la diferencia entre mayúsculas y minúsculas.
-lname {patrón} buscamos enlaces simbólicos que coincidan con el patrón.
-ilname ignora la diferencia entre mayúsculas y minúsculas.
-type busca por tipo de archivo según la siguiente tabla:

- F Archivo regular u ordinario
- D Directorio
- B Archivo de dispositivo de bloque
- C Archivo de dispositivo de carácter
- L Enlace simbólico
- p Conector o pipe con nombre

-newer busca según el contenido más reciente que un archivo dado.
-inum busca por número de i-nodo.
-emtpy busca archivos vacios.
-maxdepth n. busca de forma recursiva hasta un máximo de n niveles de subdirectorios por debajo del especificado.
-mindepth n. similar a maxdepth pero comenzará a buscar a partir de n niveles.
-group {nombre} busca por el nombre del grupo.
-perm {modo} buscamos los archivos que coincidan exactamente con los permisos en representación octal (de 0 a 7) o en representación simbólica (r para lectura, x para ejecución, w para escritura).
-links n busca por cantidad n de enlaces.
-sizen[ckMG] busca por tamaño de archivo según las siguientes unidades de almacenamiento

- c= bytes.
- k= kilobytes.
- M= Megabytes.
- G= GigaBytes.

-amin {n} búsqueda de archivos que han sido leídos hace n minutos.
-atime {n} búsqueda de archivos que han sido leídos por última vez hace nx24 horas.
-cmin {n} búsqueda de archivos cuyos permisos se han cambiado hace n minutos.
-mtime busca los archivos modificados en los últimos n días.
-uid {n} busca archivos cuyo propietario tenga el uid especificado.
-user {usuario} busca archivo cuyo propietario del mismo es el usuario.

Las opciones que emplean un argumento entero **n**, por ejemplo tamaño, fecha, hora, usan:

- +n coincide con alguna propiedad de archivo cuyo valor es mayor que n.
- -n coincide con alguna propiedad de archivo cuyo valor es menor que n.
- n coincide con alguna propiedad de archivo cuyo valor es igual a n.

Ejemplo 1

```
$find  /dev  ../sistemas /proc/1175  -name deudores2> /dev/null
```

En este ejemplo el comando busca el archivo *deudores* en los tres directorios tomados como path, y en todos sus subdirectorios. Es posible desviar la salida con error (es el caso del mensaje `acceso denegado` a ciertos directorios) indicada con el número 2 (stderr) al archivo /dev/null, para que no aparezcan por pantalla.

Ejemplo 2

```
$find  /home/rgarcia  -name  deudores
```

Entrega un listado con la ruta absoluta de los archivos *deudores* que encuentre. La búsqueda se realiza en directorio de conexión *rgarcia* y en los niveles inferiores a éste.

El patrón o expresión regular incluye un conjunto de cadenas de caracteres, es decir, no buscar un determinado archivo, sino un grupo seleccionado de ellos que tengan en común ciertos caracteres de su nombre o su extensión. El patrón debe indicarse entre comillas simples. Podemos usar los siguientes símbolos:

- ***** engloba cualquier o ningún carácter.
- **?** cualquier carácter sencillo, justamente uno.
- **[cadena]** coincidencia exactamente en uno de los caracteres contenidos en la cadena. También la cadena puede contener rangos como por ejemplo incluir todas las letras minúsculas y los números [a-z0-9].
- **[^cadena]** no coincidencia con ninguno de los caracteres de la cadena. Es la negación de [cadena].

Es aconsejable que el patrón vaya ente comillas simples.

Ejemplos de uso de expresiones regulares

`'?finan*'`	podrían ser 1finanzas y zfinan2017.
`'*.jpg'`	cualquier archivo con extensión jpg.
`'*2*.txt'`	cualquier archivo .txt que su nombre tuviera un 2.
`'*[2]*.txt`	igual que el ejemplo anterior.
`'[0-9]?sop*.??q'`	coincidirían por ejemplo los archivos 03sop.44q y 1tsop2kn.exq

Acciones:

-delete borra los archivos encontrados
-exec comando {} \; ejecutar una orden sobre el resultado de la búsqueda. Utilizamos esta acción seguida de un espacio y el comando a ejecutar sobre el valor devuelto por *find* y después {} \;
La cadena **{}** se reemplaza por los archivo(s) encontrado(s).

Ejemplo 3

```
$find  /home/rgarcia  -name  'deudo*'
```

Busca archivos cuyo nombre comiencen con la cadena *deudo.* En la salida puede mostrar por ejemplo los siguientes nombres: *deudores, deudor-05, deudor-orl*

Ejemplo 4

```
$find  .  -name '*deudor*'-name'*01'
```

En este caso los nombres archivos deben cumplir dos condiciones para que sean seleccionados:

- el nombre del archivo base debe contener la palabra deudor
- el nombre del archivo base debe terminar en 01

Ejemplo 5

```
$find / -name '[0-9]*'
```

Muestra la ruta a los archivos cuyo nombre empiece con un dígito. La busqueda se realiza en todo el sistema de archivos.

Ejemplo 6

```
$find / -name '[Mm]*'
```

Busca archivos cuyo nombre empiece con un la letra M o m.

Ejemplo 7

```
$find / -name '[a-m]*.txt'
```

Busca archivos cuyo nombre empiece con una letra entre a y m y termine en ".txt".

Ejemplo 8

```
$find  ~/  -atime 5
```

Es posible utilizar, en el *pathname*, *~/* para indicarle a *find* que la búsqueda se realizará en nuestro directorio de conexión. En este caso, el comando muestra un listado de los archivos leídos durante los últimos 5 días.

Ejemplo 9

```
$find  / -inum  102802
```

Busca todos los archivos que tienen asignado el i-nodo indicado. Los enlaces duros permiten que diferentes nombres simbólicos de archivos tengan el mismo número de nodo-i. Sin embargo, debe verificar que los archivos encontrados pertenezcan al mismo sistema de archivos, pues cada sistema de archivos tiene su propio conjunto de nodos-i. Dos archivos de diferentes sistemas de archivos con el mismo número de nodo-i no son el mismo archivo, porque hacen referencia a diferentes conjuntos de nodo-i y de bloques de datos.

Ejemplo 10

```
$find  /etc -type l
```

Busca en el directorio /etc los archivos tipo enlace y muestra un listado de los mismos.

Ejemplo 11

```
$find /home/garcia –newer /home/garcia/back-seg.tar
```

Busca según el contenido más reciente que un archivo dado en la opción es *–newer*. Este muestra la ruta de acceso a los archivos modificados después del archivo back-seg.tar.

Ejemplo 12

```
$find  /   -size +30M –mtime  -10  2> /dev/null
```

Busca los archivos de todo el File System que tienen más de 30MB y que se modificaron en los últimos 10 días.

Ejemplo 13

```
$find /var -size +3000k -exec ls -lh {} \;
```

Busca todos los archivos mayores a 3MB en /var y además muestra su salida en formato `ls`.

Ejemplo 14

```
$find /tmp -size -1024k -delete
```

Busca en el directorio /tmp los archivos menores a 1 KB y los borra.

Ejemplo 15

```
$find . -perm -u=rwx,go=r
```

Busca a partir del directorio activo y en ramas inferiores, archivos con permiso para el dueño (lectura, escritura, ejecución), para el grupo y otros (lectura).

Ejemplo 16

```
$find . -perm 744
```

Ídem al anterior

sort Ordena alfabéticamente las líneas de uno o más archivos que se indican como argumento y muestra las líneas ordenadas por la salida estándar.

Este comando no modifica el archivo original, sino que genera una salida ordenada del mismo. En realidad el ordenamiento no es estrictamente alfabético, sino más bien de acuerdo al valor ASCII de los caracteres. Los espacios en blanco son tomados por defecto como separadores de campo.

Sintaxis:

```
sort [- opciones] [archivos a ordenar...]
```

Sort supone que cada línea del texto está compuesta por campos o palabras. Por defecto el ordenamiento se realiza según el primer campo. Las líneas que empiezan con una letra mayúscula se ordenan al principio del listado, pues las mayúsculas tienen un valor ASCII menor a las letras

minúsculas. Los valores más pequeños son para el espacio y el signo !. También tiene varias opciones para controlar el orden de clasificación.

Opciones:

-r invierte el orden normal.
-n clasifica según el valor numérico (no ASCII).
-nr clasifica en orden numérico inverso.
-f no discrimina entre mayúsculas y minúsculas.
-u clasifica y elimina las líneas que son idénticas.
-k n1,[n2] Especifica un campo como clave de ordenación, comienza en n1 y acaba en n2; los números de campo empiezan en 1
-t indica el carácter utilizado como separador de campos.

Para nuestros ejemplos, suponemos creado el archivo *lista* con las siguientes líneas (apellido, legajo, documento, edad):

```
Gutierrez   45378       30569025    20
Torres      42963       32987026    18
Albama      40258       32025147    19
Kirmat      45147       33589632    16
```

Ejemplo 1

```
$sort  lista
Albama      40258       32025147    19
Gutierrez   45378       30569025    20
Kirmat      45147       33589632    16
Torres      42963       32987026    18
```

Vemos que el comando nos muestra las líneas del archivo ordenadas según el primer campo, es decir según el apellido.

Ejemplo 2

Ahora bien, si deseamos ordenar el archivo según el campo legajo, tendremos que indicarle a *sort* que vamos a ordenar por el segundo campo, pero como el número de espacios entre el apellido y el legajo varía según la línea, vamos a utilizar la opción *–b* para indicarle al comando que considere un grupo de espacios como un único espacio.

```
$sort -k2n -b lista
Albama      40258       32025147  19
Torres      42963       32987026  18
Kirmat      45147       33589632  16
Gutierrez   45378       30569025  20
```

Ordena el archivo lista por el segundo campo y lo hace según su valor numérico. Como el número de espacios entre el apellido y el legajo varía según la línea, utilizamos la opción *–b* para indicarle al comando que considere un grupo de espacios como uno solo.

Ejemplo 3

La opción **n** trata también números negativos

```
$sort -k2n -b pedido
Mirta       -30   harina
Pedro       -2.5  yerba
Susana      1.25  azucar
Esther      12    pan
```

Find ordena el archivo por el 2° campo de menor a mayor, considerando los números positivos y negativos:

Ejemplo 4

```
$ls  -l  | sort -k4
-rw-r—r--   1   Pedro   docent     25   Feb   1   11:33    rep01
-rw-r—r--   1   Mirta   docent   1972   Feb   3   15:03    rep-02
```

Muestra la salida del comando **ls** ordenada por el 4to campo, grupo al que pertenece el archivo.

Ejemplo 5

```
$cat /etc/passwd | sort -t: -k3 -f | cut -d':' -f6
```

Ordena el archivo *passwd* según el campo 3 (uid= identificador de usuario), pero sólo muestra la columna 6 (directorio de conexión).

WC	permite contar líneas, palabras y caracteres de una entrada, sea ésta la entrada estándar o un grupo de archivos, y entrega un informe por la salida estándar.

Sintaxis:

```
wc  ↙-opción ↘ archivos ...
```

wc por omisión realiza las tres operaciones: contar líneas, palabras y caracteres; y en ese orden los muestra.

Opciones:

- **- l** cuenta sólo las líneas del archivo de entrada.
- **- w** cuenta sólo las palabras del archivo de entrada.
- **-c** cuenta sólo los caracteres del archivo de entrada.

Este comando es muy simple, pero es de gran utilidad en combinación con otros comandos. Por ejemplo, supongamos que necesitamos saber cuántos archivos (incluyendo los directorios) tiene nuestro directorio, tal caso podemos ejecutar el siguiente comando:

```
$ls |wc -l
```

Es posible también darle a wc varios argumentos, por ejemplo que cuente las líneas de tres archivos:

```
$wc -l dato docu.txt practico

      300   dato
      4500  docu.txt
      963   practico
```

Actividad 1

1. Crear el archivo *directorios* con la salida del comando *ls –la*, ejecutado en su directorio de login.

```
ls -la  >directorios
```

2. Verificar que el archivo se haya creado.

```
ls | grep directorios
```

3. Mostrar todas las líneas del archivo directorios que comienzan con el carácter "-", es decir las líneas que correspondan a archivos comunes.

```
grep  '^-' directorios
```

Utilizamos el ancla ^, para que la búsqueda se realice en el primer campo, tipo de archivo y permisos.

4. Mostrar los permisos de los archivos de su directorio en cuyo nombre tengan la cadena *nome*.

```
ls -la|grep 'nome$'
```

Utilizamos el ancla $, para que la búsqueda se realice en el último campo, nombre de archivo.

5. Mostrar todos los archivos cuyos permisos para el dueño sean *rw–*.

```
ls -la|grep  '^.rw-'
```

Utilizamos el ancla ^, para que la búsqueda se realice en el primer campo, tipo de archivo y permisos. El punto, reemplaza a cualquier carácter que esté en esa posición, es decir, el tipo de archivo puede ser d, –, l, c, b; luego colocamos los tres caracteres que buscamos en esa posición, *rw–.*

Actividad 2

1. Ordenar el archivo *directorios* según el 2° campo, el contador de enlaces, y guardar la salida del comando en el archivo *dir1*.

```
sort  -k2n directorios  >  dir1
```

Si ejecutamos el comando *more dir1*, veremos que los archivos están ordenados según el numero de enlaces, primero los que tienen un enlace, luego los que tienen dos, etc.

2. Ejecutar el comando del punto anterior, pero el orden de clasificación debe ser el inverso. Guardar la salida del comando en el archivo *dir2*.

```
sort  -k2nr directorios  >  dir2
```

La opción –r invierte el orden de clasificación, la opción –n hace referencia a que el campo sobre el cual estamos trabajando, es numérico. Si vemos el contenido de dir2, los archivos estarán ordenados según el contador de enlaces, primero, por ejemplo, los que tienen cinco enlaces, luego los que tienen cuatro, etc.

3. Ordenar el archivo *directorios* según el nombre de los archivos. Redireccionar la salida del comando al archivo *dir3*.

```
sort  -k9  directorios > dir3
```

Los archivos estarán ordenados alfabéticamente según el último campo.

4. Idem al anterior, pero ordene en orden inverso de clasificación. Guarde la salida en el archivo *dir4*.

```
sort  -k9r directorios > dir4
```

5. Contar las líneas del archivo *directorios.*

```
wc  -l  directorios
```

6. Indicar la cantidad de archivos tipo enlace que existen en su directorio activo

```
ls -la | grep '^l' |wc -l
```

Actividad 3

1. En su directorio de conexión crear los subdirectorios *tareas* y *practico*.

```
cd          #vamos a nuestro directorio de conexión
mkdir tareas  practico
```

2. Copiar los archivos *dir1* y *dir2* al subdirectorio *tareas*.

```
cp  dir1  dir2  tareas

Verificamos si los archivos fueron copiados, tipeando: ls tareas
```

3. Copiar los archivos *dir3* y *dir4* al subdirectorio practico.

```
cp dir3  dir4  practico
```

4. Establecer como directorio activo a *practico*.

```
cd  practico
```

5. Buscar desde su directorio de *login*, el archivo *dir1*.

```
find  ~/   -name  dir1

Observe que el comando muestra un listado de los directorios en donde encontró
el archivo dir1. Muestra la ruta absoluta a cada uno de los archivos dir1.
La búsqueda comenzó desde el directorio de conexión, hacia las ramas
inferiores del árbol de directorios.
```

6. Buscar desde su directorio de *login*, todos los archivos cuyo nombre comience con la cadena dir.

```
find  ~/  -name  'dir*'
```

7. Buscar en su árbol de directorios, todos los archivos creados el día de hoy.

```
find  ~/ -atime 0
```

8. Buscar y mostrar en su árbol de directorios, todos los archivos directorios.

```
find  ~/  -type  d
```

Actividad 4

1. Generar un archivo con información sobre los usuarios conectados al sistema, pero no se incluya en el informe.

```
who | grep -v $LOGNAME > usuarios
```

2. Mostrar el contenido del archivo ordenado según el nombre de usuario, y si éste tiene más de una terminal abierta, ordenar de acuerdo a éste otro campo.

```
sort -k1,2 usuarios
```

3. Generar un listado de los archivos cuyo número de enlaces sea superior a 3, considere todos los archivo de /home.

```
find /home –links +3  2> /dev/null
```

4. Generar un listado ordenado de menor a mayor del campo tamaño de archivo y nombre de archivo. Trabaje sobre el contenido del directorio de conexión.

```
ls –l ~/ |tr –s ' ' |cut –d ' ' –f5,9 |sort –n | more
```

Capítulo 7

Seguridad y Protección

Seguridad Informática

La seguridad informática se define como cualquier medida que impida la ejecución de operaciones no autorizadas sobre un sistema o red informática, cuyos efectos pueden causar daños sobre la información, comprometer su confidencialidad, autenticidad o integridad, disminuir el rendimiento de los equipos o bloquear el acceso de usuarios autorizados al sistema.

Seguridad del sistema de archivos

En cualquier sistema multiusuario, los sistemas de archivos contienen información muy valiosa por lo que es preciso que existan métodos que protejan los datos, por ejemplo que impidan a un usuario no autorizado copiar, borrar, modificar algún archivo sobre el cual no tiene permiso.

Para esto Linux, utiliza un *sistema de autenticación de usuarios* al iniciar una sesión y un *sistema de permisos* de acceso a los archivos y directorios. Para tener en cuenta, autenticar es asegurar que una entidad (usuario) es quien dice ser.

Autenticación de usuarios

Comienza cuando se ejecuta el comando *login* que es el que pide el nombre de usuario con el cual el sistema operativo identifica una cuenta de usuario. Luego pide la palabra clave (password), la cual no se visualiza en pantalla. *Login* toma el *username* y verifica su existencia en el archivo */etc/password*. Así mismo obtiene del archivo */etc/group* el grupo al que pertenece el usuario.

Utiliza un sistema de contraseñas shadow (sombra) que mejora la seguridad del sistema al mover las contraseñas encriptadas desde el archivo */etc/passwd* que puede leer todo el mundo, a */etc/shadow*, el cual sólo puede ser leído por el usuario root que es el único que posee las claves para desencriptar ese archivo. También almacena información sobre la vigencia de las contraseñas.

Una vez que el sistema inicia la sesión, ubica al usuario en un directorio de trabajo que le es propio, donde él es el dueño, generalmente */home/username*.

Cada usuario puede cambiar su propia clave cuando lo desee, y root (supervisor) tiene el poder de cambiar la clave de todos los usuarios. Para ello existe un comando:

passwd	este comando se utiliza para realizar el cambio de clave

Sintaxis:

```
passwd usuario [enter]
```

Obtendrá la siguiente salida:

```
Changing password for usuario
(current) Unix Password:
New Unix Password:
Retype New Unix Password:
```

En donde deberá ingresar la clave actual y luego ingresar dos veces la nueva clave.

Sistema de Permisos r w x

El otro método que utiliza Linux para brindar mayor seguridad es mantener **un sistema de permisos** de acceso a los archivos muy estricto, a fin de controlar qué es lo que se puede hacer con ellos, y quien lo puede hacer. Cada archivo posee tres tipos de permisos que se identifican con letras y son:

r	Permiso de lectura del contenido del archivo, o permiso de listar el contenido en el caso de directorios.
w	Permiso de escritura y modificación del archivo, para directorios, permite crear nuevos directorios y archivos o borrar los ya existentes en dicho directorio.
x	Permiso de ejecución del archivo, si es un programa, o para directorios permite al usuario explorar dicho subdirectorio.

Al contrario que en ***Windows*** los programas ejecutables de ***Linux*** no están marcados por una determinada extensión (***.exe***) sino por un atributo, el permiso de ejecución ***x***. Si se elimina este atributo a un programa, ***Linux*** no será capaz de ejecutarlo.

A su vez cada uno de estos permisos son fijados para tres clases de usuarios:

- ***u*** el usuario o dueño del archivo (normalmente el que lo creó)
- ***g*** el grupo de usuarios al que pertenece el dueño.
- ***o*** el resto de usuarios.

Así por ejemplo, un archivo determinado puede tener permiso para ser leído, escrito y ejecutado por su dueño, leído y ejecutado por el grupo al que pertenece y no tener ningún tipo de acceso para los demás usuarios. Como se puede entender este tipo de mecanismo es especialmente útil cuando se trabaja en grupo en un determinado proyecto.

Con el comando ***ls*** y la opción ***– l*** (extendido), se mostrará el listado largo de los archivos, el cual incluye los permisos que posee cada archivo.

Ejemplo

```
$ls –l archivo

-rw-r--r-- 1 usuario1 users 505  May 3 18:30  sueldos
```

El primer campo representa tipo y permisos del archivo. El tercer campo es el propietario del archivo (*usuario1*) y el cuarto es el grupo al que pertenece el archivo (*users*). Obviamente el último campo es el nombre del archivo y los demás campos fueron analizados anteriormente.

Este archivo pertenece a *usuario1* y al grupo *users.* La cadena *-rw-r--r--* nos informa lo siguiente:

El primer carácter indica el tipo de archivo (visto anteriormente), es decir nos indica si es un archivo regular (-). Los 9 caracteres siguientes nos muestran por orden, los permisos para el propietario, para el grupo del archivo y para cualquier otro usuario.

Luego concluimos que *usuario1* tiene permisos de lectura y escritura para el archivo *sueldos* pero no de ejecución. El grupo *users* tiene solo permisos de lectura (r) al igual que los otros usuarios.

Ejemplos

-rwxr-xr-x	*El propietario del archivo puede leer, escribir y ejecutar el archivo. Los usuarios pertenecientes al grupo del archivo, y todos los demás usuarios pueden leer y ejecutar.*
-rw-------	*El propietario del archivo puede leer y escribir. Nadie más puede acceder al archivo.*
-rwxrwxrwx	*Todos los usuarios pueden leer, escribir y ejecutar el archivo.*

Los permisos otorgados a un archivo dependen de los permisos del directorio en el cual el documento está localizado: para ser capaz de leer un archivo, el usuario necesita tener el permiso de lectura para el archivo y el permiso de ejecución para el directorio que lo contiene.

Por lo tanto, si un usuario determinado no quiere que nadie más vea sus archivos, puede lograr esto eliminando los permisos de ejecución de su directorio personal para todos los demás usuarios. De esta manera, sólo él (y por supuesto, el administrador "root") podrán leer cualquiera de sus archivos, sin importar cuáles sean los permisos individuales de los archivos.

Por ejemplo, aunque un archivo tenga los permisos -rwxrwxrwx, otros usuarios no podrán acceder a él a menos que también tengan permiso de lectura y ejecución para el directorio en el cual se encuentra el archivo. Es decir, si María quiere restringir el acceso a todos sus archivos, podría simplemente poner los permisos a su directorio de trabajo */home/maría* de la siguiente forma:

```
drwx------  /home/maria
```

Sintetizando, para acceder a un archivo, se debe de tener permiso de ejecución de todos los directorios a lo largo del camino de acceso al archivo, además de permiso de lectura (o ejecución) del archivo en particular.

Cambio de permisos

chmod	Este comando se usa para modificar los permisos de un archivo.

Los permisos de acceso a un archivo solo los fija el propietario del archivo y no el administrador del sistema. Este solo es responsable de sus archivos y los del sistema.

Este comando tiene tres opciones, llamados modos:

Modo simbólico

Sintaxis:

```
chmod {u,g,o,a}{+,-}{r,w,x} nombre_del_archivo
```

Entre llaves indicamos las opciones de las modificaciones a realizar sobre el archivo.
En el primer grupo se especifica a qué usuarios se está otorgando el permiso:

u = (user) propietario del archivo
g = (group) grupo al que está asociado el archivo
o = (other) resto de los usuarios que no son propietarios ni pertenecen al grupo
a = (all) todos los usuarios del sistema

En el caso que no se especifique se toma por defecto a (all).

En el segundo grupo de opciones se especifica si se están añadiendo permisos (+) o quitándolos (-). Finalmente se especifica el tipo de permiso o sea read, write o execute.

Ejemplos

$chmod a+r notas al igual que *chmod +r notas*

Da a todos los usuarios permiso de lectura al archivo notas.

$chmod og-x archivo

Quita permisos de ejecución a todos los usuarios excepto al propietario.

$chmod ug+rx,o-w archivo

A user y group agrega permisos de lectura y ejecución. A other le quita permiso de escritura.

Modo absoluto

Sintaxis:

```
chmod u={r,w,x,-},g={r,w,x,-},o={r,w,x,-}  nombre_del_archivo
```

En el modo absoluto a cada grupo de usuarios se le asigna con el signo igual el o los permisos que tendrá el archivo independiente de los permisos que posea anteriormente. Por lo que es necesario definir todos los permisos deseados.

Ejemplos

$chmod u=rwx,g=rx,o=w archivo

User adquiere los permisos rwx, group adquiere solamente r y x, other adquiere solamente w.

$chmod u=rwx,g=w archivo

User adquiere los permisos rwx, group adquiere solamente w, other no se modifica.

Modo numérico

Es el tercer modo de este comando, en el cual se especifican los permisos mediante ***notación numérica octal (0-7)*** de 4 dígitos en donde el primer digito está reservado para los bits suid, sgid y sticky, el segundo dígito para los permisos del propietario, el tercer dígito para los permisos del grupo, y el cuarto dígito para el resto de los usuarios. De esta manera, un archivo tiene tres números asignados: propietario, grupo y otros.

Cada permiso lleva “asociado” un número:

			- r w x	
	4	para lectura	0 1 0 0	
+	2	para escritura	0 0 1 0	en binario
	1	para ejecución	0 0 0 1	
	0	ausencia de permiso	0 0 0 0	
	7		0 1 1 1	

Para cada clase de usuario se calcula la suma de permisos que se desea conceder, el número resultante será el parámetro utilizado por ***chmod*** para modificar los permisos del archivo.

Ejemplo 1

```
$chmod 0754 archivo
```

El 7 implica los tres permisos otorgados a la vez para el propietario (0111, los bits *r w x* activados).
El 5 implica lectura y ejecución para el grupo (0101, los bits *r - x* activados).
El 4 implica lectura solamente para los otros usuarios (0100, el bit *r - -* solamente activado).

Ejemplo 2

Si se desea otorgar permiso de lectura y escritura al propietario del archivo, permiso de lectura al grupo y ningún permiso al resto de los usuarios, el número que habrá que utilizar es 640.

```
$chmod 0640 archivo
```

En el entorno gráfico se pueden ver los permisos utilizando el Gestor de Archivos de GNOME, presionando el botón derecho del ratón en el archivo, y escogiendo propiedades desplegables, y entonces la pestaña Permisos. Utilizando este diálogo, se pueden también cambiar los permisos sólo presionando en el cuadrado que representa el permiso para modificar su estado.
Por supuesto, sólo el dueño del archivo o el administrador del sistema puede cambiar los permisos de un archivo.

Los bits especiales suid, sgid y sticky

Como ya vimos, los permisos de los archivos en Linux se corresponden con un número en octal que varían entre 000 y 777; sin embargo, existen unos permisos especiales que hacen variar ese número entre 0000 y 7777, se trata de los bits de permanencia ó sticky (1000), sgid (2000) y suid (4000).

El bit de suid o *setuid* se activa sobre un archivo, añadiéndole 4000 a la representación octal de los permisos del archivo, otorgándole además permiso de ejecución con los permisos del propietario, cualquiera sea el usuario que lo ejecuta; al hacer esto, en lugar de la x en la primera terna de los permisos, aparecerá una ***s***, o una ***S*** si no hemos otorgado el permiso de ejecución correspondiente (en este caso el bit no tiene efecto).

Ejemplos

```
#chmod 4777 archivo1
#chmod 4444 archivo2
#ls -l archivo1
-rwsrwxrwx 1 root other  0  May 9  17:51  archivo1
#ls -l archivo2
-r-Sr--r--  1 root  other  0  May 9  17:51 archivo2
```

El bit *suid* activado sobre un archivo indica que todo aquel que ejecute el archivo va a tener durante la ejecución los mismos privilegios que quien lo creó; es decir que indica que el usuario “toma prestada" la identificación del dueño mientras ejecuta el archivo.

Todo lo que acabamos de comentar con respecto al bit *setuid* es aplicable al bit *setgid* pero a nivel de grupo del archivo en lugar de propietario. Todo usuario que ejecute un programa *setgidado* tendrá los privilegios del grupo al que pertenece el archivo.

Para activar el bit de *setgid* sumaremos 2000 a la representación octal del permiso del archivo y además tendremos que darle permiso de ejecución a la terna de grupo; si lo hacemos, la ***s*** o ***S*** aparecerá en lugar de la x en esta terna.

Si el archivo es un directorio y no un archivo plano, el bit *setgid* afecta a los archivos y subdirectorios que se crean en él, estos tendrán como grupo propietario al mismo que el directorio *setgidado*, siempre que el proceso que los cree pertenezca a dicho grupo.

Los bits de *setuid* y *setgid* dan a Linux una gran flexibilidad, pero constituyen también la mayor fuente de ataques realizados por usuarios internos al sistema, con el objetivo de aumentar su nivel de privilegio. Esto afecta a la seguridad del sistema.

Ejemplos

```
#chmod 2777 archivo1
#chmod 2764 archivo2
#ls -l archivo1
 -rwxrwsrwx  1  root  other  0  May 9  17:51    archivo1
#ls -l archivo2
 -rwxrwSr--  1 root   other  0  May 9  17:51    archivo2
```

Por otra parte, el *sticky bit* o bit de permanencia se activa sumándole 1000 a la representación octal de los permisos de un determinado archivo y otorgándole además permiso de ejecución; si hacemos esto, veremos que en lugar de una x en la terna correspondiente al resto de usuarios aparece una ***t*** (si no le hemos dado permiso de ejecución al archivo, aparecerá una ***T***).

```
#chmod 1777 archivo1
#chmod 1774 archivo2
#ls -l archivo1
 -rwxrwxrwt 1      root  other 0  May 9  17:51   archivo1
#ls -l archivo2
 -rwxrwxr-T 1     root  other  0  May 9  17:51  archivo2
```

El bit sticky activado le indica al sistema operativo que se trata de un archivo muy utilizado, por lo que es conveniente que permanezca en memoria principal el mayor tiempo posible; esta opción se utilizaba en sistemas antiguos que disponían de muy poca RAM, pero hoy en día prácticamente no se utiliza. Lo que sí sigue vigente, es el uso del *sticky bit* activado sobre un directorio: en este caso se indica al sistema operativo que, aunque los permisos 'normales' digan que cualquier usuario pueda crear y eliminar archivos (por ejemplo, un 777 octal), sólo el propietario del archivo y el administrador pueden borrar un archivo guardado en un directorio con estas características.

Aunque cualquier usuario puede hacer que aparezca una **t** o una **T** en sus archivos o directorios, este bit sólo tiene efecto cuando es activado por el administrador (root), se utiliza principalmente en directorios del sistema de archivos en los que interesa que todos puedan escribir pero que no todos puedan borrar los datos escritos, como /tmp/ o /var/tmp/

Si en lugar de especificar el valor octal de los permisos queremos utilizar la forma simbólica de *chmod*, utilizaremos +t para activar el bit de permanencia, g+s para activar el de *setgid* y u+s para hacer lo mismo con el de *setuid*; si queremos resetearlos, utilizamos un signo '-' en lugar de un '+' en la línea de comandos.

Atributos de los archivos

En los sistemas de archivos ext2, ext3 y ext4 (Extended File System) de Linux también existen ciertos atributos para los archivos que pueden ayudar a incrementar la seguridad de un sistema. Los atributos de los archivos en ext4 son los mostrados en la siguiente tabla.

Atributo *Significado*

- **A** Establece que la fecha del último acceso (atime) no se modifica.
- **a** Establece que el archivo sólo se puede abrir en modo de adjuntar para escritura.
- **c** Establece que el archivo es comprimido automáticamente en el disco por el núcleo del sistema operativo. Al realizar la lectura de este archivo, se descomprimen los datos. La escritura de dicho archivo comprime los datos antes de almacenarlos en el disco.
- **D** Cuando se trata de un directorio, establece que los datos se escriben de forma sincrónica en el disco. Es decir, los datos se escriben inmediatamente en lugar de esperar la operación correspondiente del sistema operativo. Es equivalente a la opción dirsync de mount, pero aplicada a un subconjunto de archivos.
- **d** Establece que el archivo no sea candidato para respaldo al utilizar la herramienta dump.
- **e** Indica que el archivo o directorio utiliza extensiones (extents) para la cartografía de bloques en la unidad de almacenamiento, particularmente de sistemas de archivos Ext4. Cabe señalar que chattr es incapaz de eliminar este atributo.
- **i** Establece que el archivo será inmutable. Es decir, se impide que el archivo sea eliminado, renombrado, que se pueden apuntar enlaces simbólicos hacia éste o escribir datos en el archivo.
- **j** En los sistemas de archivos ext3 y ext4, cuando se montan con las opciones data=ordered o data=writeback, se establece que el archivo será escrito en el registro por diario (Journal). Si el sistema de archivos se monta con la opción data=journal (opción predeterminada), todo el sistema de archivos se escribe en el registro por diario y por lo tanto el atributo no tiene efecto.
- **s** Cuando un archivo tiene este atributo, los bloques utilizados en el disco duro son escritos con ceros, de modo que los datos no se puedan recuperar por medio alguno. Es la forma más segura de eliminar datos.
- **S** Cuando el archivo tiene este atributo, sus cambios son escritos de forma sincrónica en el disco duro. Es decir, los datos se escriben inmediatamente en lugar de esperar la operación correspondiente del sistema operativo. Es equivalente a la opción sync de mount.
- **u** Cuando un archivo con este atributo es eliminado, sus contenidos son guardados permitiendo recuperar el archivo con herramientas para tal fin.

No todos estos atributos nos interesan para el tema de la seguridad. Uno de los atributos interesantes es "**a**"; este atributo sobre un archivo indica que sólo se puede abrir en modo escritura para añadir datos, pero nunca para eliminarlos. Tan importante es, que sólo el administrador tiene el privilegio suficiente para activarlo o desactivarlo. De esta manera cuando un intruso consigue entrar al sistema, no puede borrar sus huellas.

Otro atributo del sistema de archivos es "i"' (archivo inmutable); un archivo con este atributo activado no se puede modificar de ninguna forma, (ni añadir datos ni borrar, ni eliminar el archivo, ni enlazarlo mediante ln). También es privilegio del administrador activar o desactivar el atributo "i" de un archivo.

Atributos interesantes, aunque menos importantes que los anteriores, son `s' y `S'. Si borramos un archivo con el atributo `s' activo, el sistema va a rellenar sus bloques con ceros en lugar de efectuar un simple unlink(), para así dificultar la tarea de un atacante que intente recuperarlo.

Por su parte, el atributo `S' sobre un archivo hace que los cambios sobre el archivo se escriban inmediatamente en el disco en lugar de esperar el sync del sistema operativo.

lsattr	comando que lista o visualiza los atributos de un archivo

Veamos ahora como visualizar el estado de los diferentes atributos, utilizaremos lsattr y el nombre del archivo, cuya salida indicará con la letra correspondiente cada atributo del archivo o un signo '-' en el caso de que el atributo no esté activado.

Ejemplo

```
#lsattr  /tmp/archivo
s--S-a-- /tmp/archivo
```

Dentro de Linux todo es un archivo, desde la memoria física del equipo, hasta el ratón, pasando por módems, teclado, impresoras o terminales. Esta filosofía de diseño es uno de los factores que más éxito y potencia le proporciona, pero también uno de los que más peligros genera, un simple error en un permiso puede permitir a un usuario modificar todo un disco duro o leer los datos tecleados desde una terminal.

Por esto, una correcta utilización de los permisos, atributos y otros controles sobre los archivos es vital para la seguridad de un sistema.

chattr	comando que modifica un atributo de un archivo

Para modificar un atributo utilizamos el comando *chattr,* que recibe como parámetros el nombre del atributo junto a un signo `+' o `-', en función de si deseamos activar o desactivar el atributo, y también el nombre de archivo correspondiente.

Sintaxis:

```
chattr [-RV] +-=[AacDdijsSu] [-v versión] archivos
```

Ejemplos

```
# lsattr /tmp/archivo
-------- /tmp/archivo
# chattr +a /tmp/archivo
# chattr +Ss /tmp/archivo
# lsattr /tmp/archivo
s--S-a-- /tmp/archivo

# chattr -sa /tmp/archivo
# lsattr /tmp/archivo
---S---- /tmp/archivo
```

Actividad 1

1. El comando *login* verifica el *username* y la clave de acceso. ¿En qué archivo hace esta verificación?

2. ¿Un usuario puede cambiar su clave de acceso? ¿Qué comando usaría?

3. ¿Qué tamaño mínimo debe tener una clave?

4. ¿Qué usuario puede cambiar la clave de otros usuarios? ¿Con qué comando?

5. ¿Cuáles son las operaciones que puede realizar un usuario sobre un archivo si los permisos de este son: **–r-x-w---x**?

6. ¿Qué usuarios tienen autorización para modificar los permisos de un determinado archivo?

7. ¿Qué operación/es podemos realizar sobre un directorio si éste tiene el permiso de ejecución asignado?

Actividad 2

1. Comenzar la sesión de trabajo

```
login:
password:
```

2. Cambie su clave de acceso actual y coloque una clave autorizada

```
passwd
```

3. Luego abra una sesión en otra consola virtual en modo texto

```
Ctrl+Alt+F2
Login:   alumno
Password: .............. (ingrese la nueva clave)
```

4. Visualice los permisos otorgados a los siguientes archivos y explíquelos.

/etc/passwd : _
/etc/shadow : _
/bin/login : _
/bin/ls : _
/etc/hosts : _

5. Cree un archivo llamado *copiaclave* que contenga el archivo */etc/shadow y* verifique los permisos que posee y compare.

```
cat  /etc/shadow > copiaclave
ls –l copiaclave
```

6. Cambie los permisos del archivo creado *(copiaclave*) usando el modo simbólico, otorgándole permisos de lectura <u>solamente</u> al propietario, al grupo y a los demás usuarios. Verifique los cambios realizados.

```
chmod ugo+r-wx copiaclave
chmod a+r-wx copiaclave
ls –l copiaclave
```

Deberá obtener los siguientes permisos: r - - r - - r - -

7. Haga una copia del archivo *copiaclave* llamado *copiaclave2* y usando el modo absoluto modifique los permisos del nuevo archivo otorgándole derechos de lectura y ejecución al propietario, solamente de lectura al grupo y ningún derecho al resto. Muestre los permisos modificados.

```
cp copiaclave copiaclave2
chmod u=rx,g=r,o=-  copiaclave2
ls –l copiaclave2
```

Deberá obtener los siguientes permisos: r - x r - - - - -

8. Otorgue los mismos derechos del punto anterior al archivo *copiaclave,* pero utilice el modo numérico (octal).

```
chmod  540 copiaclave
ls –l copiaclave
```

9. Cree un subdirectorio llamado "prueba". Compruebe los derechos que se le otorgaron.

```
mkdir prueba
ls –l | grep prueba
```

10. Modifique los permisos del directorio *prueba* de manera que se vean como:
r- - r - - - - -, utilice cualquier método. Luego verifique los cambios con: *ls -l*

11. Cree un archivo llamado *nuevo* y otorgue permiso de ejecución con sus permisos como propietario a los demás usuarios, además de permisos de lectura para todos los usuarios (setuid). Verifique el cambio.

```
chmod 4444 nuevo
ls – l nuevo
```

Deberá obtener los siguientes permisos: r – S r - - r - -

12. Otorgue al archivo *nuevo* derecho de lectura y modificación para todos los usuarios y privilegios de ejecución del grupo al que pertenece el archivo (segid).

```
chmod 2666 nuevo
ls –l nuevo
```

Deberá obtener los permisos: r w - r w S r w

13. Modifique los atributos del archivo *nuevo* para que sólo se pueda abrir en modo escritura para añadir datos. ¿Qué usuario deberá realizar la operación? Verifique los cambios realizados.

```
Usuario: root
# chattr +a nuevo
# lsattr nuevo
```

14. Cambie los atributos del archivo *nuevo* para que no se pueda modificar de ninguna forma. Visualice los cambios.

```
# chattr +i nuevo
# lsattr Nuevo
```

Actividad 3

1. Comenzar la sesión de trabajo

```
login:
password:
```

2. Luego abra una sesión en otra consola virtual en modo texto

```
Alt+F2
Login:   alumno
Password: .............. (ingrese la nueva clave)
```

3. Visualice los permisos otorgados a los siguientes archivos y explíquelos.
/etc/crontab: _
/bin/cat : _
/usr/bin/yes: _

4. Copie el archivo */etc/crontab* en *copia y* verifique los permisos de ambos. Explique.

```
cp   /etc/crontab  copia
ls –l copia etc/crontab
```

5. Cambie los permisos del archivo *copia* usando el modo simbólico, otorgándole todos los permisos al propietario, permiso de lectura y ejecución al grupo y demás usuarios. Verifique los cambios realizados.

```
chmod u+rwx,go+rx copia
ls –l copia
```

Deberá obtener los siguientes permisos: r w x r - x r - x

6. Cree un archivo llamado *comandos* que contenga los archivos de */bin,* y verifique el contenido del archivo y los permisos que posee.

```
ls  /bin > comandos
cat comandos
ls –l comandos
```

7. Utilizando el modo absoluto modifique los permisos del archivo *comandos* otorgándole derechos de lectura y ejecución al propietario, solamente de lectura al grupo y al resto de los usuarios. Muestre los permisos modificados.

```
chmod u=rx,g=r,o=r comandos
ls –l comandos
```

Deberá obtener los siguientes permisos: r - x r - - r - -

8. Otorgue los mismos derechos del punto 5 al archivo *comandos,* pero utilice el modo numérico (octal).

```
chmod 0755 comandos
ls –l comandos
```

Deberá obtener los siguientes permisos: r w x r - x r – x

9. Realice un enlace duro del archivo *comandos* como *comandosln* y verifique los permisos de ambos.

```
ln   comandos comandosln
ls  –li comandos comandosln
```

10. Modifique los permisos del archivo *comandosln* de manera que se vean como:
r—r-----, utilice cualquier método. Luego verifique los cambio con: *ls -l*
¿Qué sucedió con los permisos de los archivos *comandos y comandosln? ¿Por qué?*

11. Realice un enlace simbólico del archivo */etc/crontab* como *crono***,** verifique los permisos de ambos y otorgue a *crono* permiso de lectura, modificación y ejecución al propietario, lectura y ejecución para el grupo y de lectura para los demás usuarios. Utilice modo numérico. ¿Qué sucedió con los permisos de */etc/crontab y crono?* ¿Por qué*?*

```
ln –s /etc/crontab crono
ls /etc/crontab crono
chmod 0744 crono
ls /etc/crontab crono
```

12. Otorgue al archivo comandos derechos de lectura para todos los usuarios y privilegios de ejecución del grupo al que pertenece el archivo (segid).

```
chmod 2444 comandos
ls -l comandos
```

Obtendrá los permisos: r - - r - S r - -

13. Modifique los atributos del archivo *crono* para que sólo se pueda abrir en modo escritura para añadir datos. Verifique los cambios realizados.

```
# chattr +a crono
# lsattr crono
```

14. Cambie los atributos del archivo *crono* para que no se pueda modificar de ninguna forma. Visualice los cambios.

```
# chattr +i crono
# lsattr crono
```

Capítulo 8

Administración de Procesos

Conceptos generales sobre Procesos

Linux es un sistema multitarea, ya que puede realizar distintas tareas concurrentemente (puede comenzar una antes de que termine otra). Todos los recursos del sistema pueden ser compartidos por muchos usuarios y el sistema operativo es el encargado de crear la ilusión de que cada usuario tiene dedicado el sistema completo para él. Sin embargo, en realidad, entre que pulsamos la tecla y se completa la orden introducida, el sistema atiende cientos de peticiones de todos los usuarios.

Un proceso, en el concepto más simple, es un programa cargado en la memoria principal destinado a ejecutarse. Pero desde un enfoque más minucioso, un proceso es una *instancia de un programa* en ejecución con su propio espacio de direcciones. Todo proceso consta de un programa ejecutable, sus datos, su pila, sus punteros de pila y programa, los registros y, en definitiva, cualquier información que sea necesaria para que este pueda ser ejecutado y tiene una existencia temporaria en la memoria de la máquina. Para que el sistema pueda gestionar los procesos, cada uno se representa por una estructura de datos *task_struct.* El vector *task* es una lista de punteros a estructuras *task_struct.*

Todo lo que ejecutamos bajo LINUX es un proceso que puede identificarse mediante un PID (Número Identificador de Proceso) que es único para cada uno de los procesos que se encuentran en ejecución. Este PID es un entero entre 0 y 65564, comenzando por 0 cuando se arranca el sistema y volviendo a empezar en 0 al llegar al máximo, pero siempre se asigna un PID que no esté asociado a un proceso activo en el momento de ejecutar el nuevo proceso. Cada proceso también tiene identificadores de usuario y grupo, que se usan para controlar el acceso de los procesos a los archivos y dispositivos del sistema.

Los procesos nuevos se clonan de un proceso existente. Cada proceso en el sistema, excepto el proceso *init,* tiene un proceso padre. Cada estructura *task_struct* mantiene punteros a su proceso padre, a sus hermanos (pues tienen el mismo padre) y a sus procesos hijos. Estas relaciones se pueden ver con el comando *pstree.*

Estados y tipos de procesos

En un sistema operativo multitarea los estados básicos en los que se puede encontrar un proceso son:

Id Estado	Estado	Descripción
S	Sleep	Suspendido. Con espera interrumpible (esperando que se complete un evento). Puede ser interrumpido por alguna señal.
D	Sleep	Suspendido. Con espera No interrumpible (generalmente esperand E/S)
T	Stopped	(Detenido) porque ha recibido una señal de detención.
R	Running	En ejecución. Solo existe un proceso utilizando la CPU que puede esta en dos modos: • modo user: ejecuta instrucciones del programa de usuario. • modo kernel: ejecuta instrucciones del kernel (llamadas al sistema operativo).
W		paginado (no válido a partir del kernel 2.6.xx)

Id Estado	Estado	Descripción
Z	Zombie	El proceso ha terminado pero no está muerto porque su estructura *task_struct* permanece referenciada desde el vector *task*. Es un proceso que aún tiene dependencias.
X	Dead	Muerto (nunca debe ser visto).

Tabla 1. Estados de los procesos en Linux

En LINUX podemos distinguir tres tipos de procesos, cada uno con características y atributos diferentes, en función de cómo haya comenzado su ejecución y son:

1. **Interactivos:** Iniciados y controlados por un shell. Pueden ejecutarse en primer plano (foreground) o segundo plano (background).
2. **En cola**: No están asociados a ningún terminal, sino que se envían a una cola en la que esperan para ejecutarse secuencialmente.
3. **Demonios**: Usualmente lanzados al iniciar el sistema y se ejecutan en background.

pstree muestra en forma jerárquica (padre-hijos) los procesos que se están ejecutando en el sistema

Muestra en forma de árbol los procesos, los que nos permite conocer las dependencias que existen entre éstos.

Sintaxis:

```
pstree [opciones] [pid|user]
```

Opciones

-p Muestra los PIDs de los procesos
-a Muestra los argumentos de la línea de comandos. Si la línea de comandos de un proceso termina, ese proceso se indica entre llaves "{ }".
-h Resalta el proceso actual y los que se encuentren sobre él.

Ejemplo

```
#pstree  -aph 3047
```

Muestra el árbol de procesos a partir del proceso de PID n° 3047, es decir, los subprocesos de éste.

```
root@debian-monica:~# pstree -aph 3047
x-terminal-emul,3047
  ├─bash,3049
  ├─bash,3055
  ├─bash,3060
  │   └─pstree,3076 -aph 3047
  ├─gnome-pty-helpe,3048
  └─{x-terminal-emul},3050
root@debian-monica:~#
```

Tabla de procesos

Linux al igual que Unix, utiliza una *tabla de procesos*, en la que se centraliza la información de cada uno de los procesos. Algunos datos de esa tabla pueden mostrarse a través del comando *ps* que obtiene información de /proc.

ps	muestra información sobre los procesos activos en el sistema.

Admite opciones de las versiones propias de **System V** (precedidas por un guión), de **BSD** (sin guiones) y de **GNU** (precedidas por dos guiones).

Según las opciones utilizadas, el comando ps proporciona la siguiente información:

Campo	Descripción
UID	El ID / nombre de usuario efectivo. Propietario del proceso.
PID	El ID de proceso.
PPID	El ID de proceso padre.
C	El uso del procesador.
CLS	La clase de programación a la que pertenece el proceso, como tiempo real, sistema o tiempo compartido. Este campo sólo se incluye con la opción -c.
%CPU	Porcentaje de tiempo que el proceso estuvo en ejecución desde que se inició.
%MEM	Porcentaje de memoria física utilizada por el proceso.
PRI	Prioridad del proceso.
NI	El número de nice del proceso, que contribuye a su prioridad de programación. Aumentar el valor del comando nice de un proceso significa reducir su prioridad.
ADDR	La dirección en memoria de la imagen del proceso.
SZ	El tamaño de la dirección virtual del proceso (imagen del proceso).
WCHAN	La dirección de un evento o bloqueo por el que el proceso está inactivo, (durmiendo o esperando)
STIME	La hora de inicio del proceso en horas, minutos y segundos.
TTY	El terminal desde el cual se inició el proceso o su proceso principal. Un signo de interrogación indica que no existe un terminal de control.
TIME	La cantidad total de tiempo de CPU utilizado por el proceso desde que comenzó.
CMD	El comando que generó el proceso.
F	Flags asociados con el proceso. Cada bit de la palabra indica una condición de tipo de proceso. **1** bifurcado pero no ejecutado. **4** tiene privilegios de root.
S ó STAT	Estado del proceso.
VSZ	Tamaño de la memoria virtual del proceso en KB
RSS	"resident set size", tamaño de la memoria física usada en KB.

En la columna STAT además del estado del proceso (Tabla 1) pueden mostrarse los siguientes códigos:

- < Tiene una prioridad mayor que lo normal
- N Tiene una prioridad menor que lo normal
- L Tiene páginas bloqueadas en memoria
- s Es un líder de sesión (se trata de un proceso que ha iniciado una nueva sesión)
- l Es multihilado
- + Está en el grupo de procesos en primer plano

Sintaxis:

```
ps [opciones]
```

Opciones:

-e	visualiza información sobre "todos" los procesos del sistema.
l	muestra información más completa sobre los procesos (difiere a poner el signo menos delante).
-f	visualiza los parámetros con los que se levantó el proceso.
-x	muestra procesos que no están controlados por ninguna terminal.
-uusuario	visualiza información de los procesos del usuario indicado.
-a	obtiene todos los procesos que estén asociados a una terminal.
-r	sólo procesos cuyo estado sea ejecutándose.
-H	muestra la jerarquía de procesos.
-txx	los procesos asociados a la terminal **xx**.
aux	muestra información adicional sobre todos los procesos que se están ejecutando en el sistema y no solo los de una terminal.

Ejemplo 1

```
$ps
PID         TIME      TTY    COMMAND
98          0.4      ?      /usr/sbin/atd
1           0.4      ?      init [2]
2           0.0      ?      (kswapd)
12          0.1      ?      update
79          0.6      ?      /sbin/syslogd
```

Sin argumentos, el comando *ps* muestra la información acerca de los procesos asociados a la sesión de un usuario. La primera columna muestra el identificador de proceso o PID. Cada proceso listado tiene un tiempo de ejecución (TIME) asociado a él, que no es tiempo real sino tiempo de CPU que el proceso ha utilizado desde que comenzó. En otra columna del listado, se presenta la terminal (TTY) del que lee su entrada el proceso y a la cual escribe. Generalmente un proceso está asociado a una terminal y algunos procesos no estarán asociados a ninguna terminal, en cuyo caso aparece en la columna TTY el carácter '?'. La última columna COMMAND muestra el comando que lanzó ese proceso.

Ejemplo 2

```
$ps -aux
```

Lo interpreta como el comando "ps aux" e imprime una advertencia. Muestra todos los procesos en el sistema en el formato orientado al usuario.x (elimina la restricción BSD "must have a tty" para agregar procesos que no tengan una tty asociada).

Ejemplo 3

```
$ps -eH
```

Muestra el árbol de procesos.

Ejemplo 4

```
$ps -e
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0  3502  3497  0  80   0 -  1830 -      pts/1    00:00:00 bash
0 R     0  5170  3502  0  80   0 -  1017 -      pts/1    00:00:00 ps
```

La opción *-e* de *ps* visualiza todos los procesos activos en la máquina y su salida es muy importante para poder diagnosticar el sistema, ver si hay procesos colgados o con dificultades, ver el tiempo que lleva ejecutándose, qué recursos utiliza, la prioridad relativa y conocer el PID.

```
$ ps -l
```

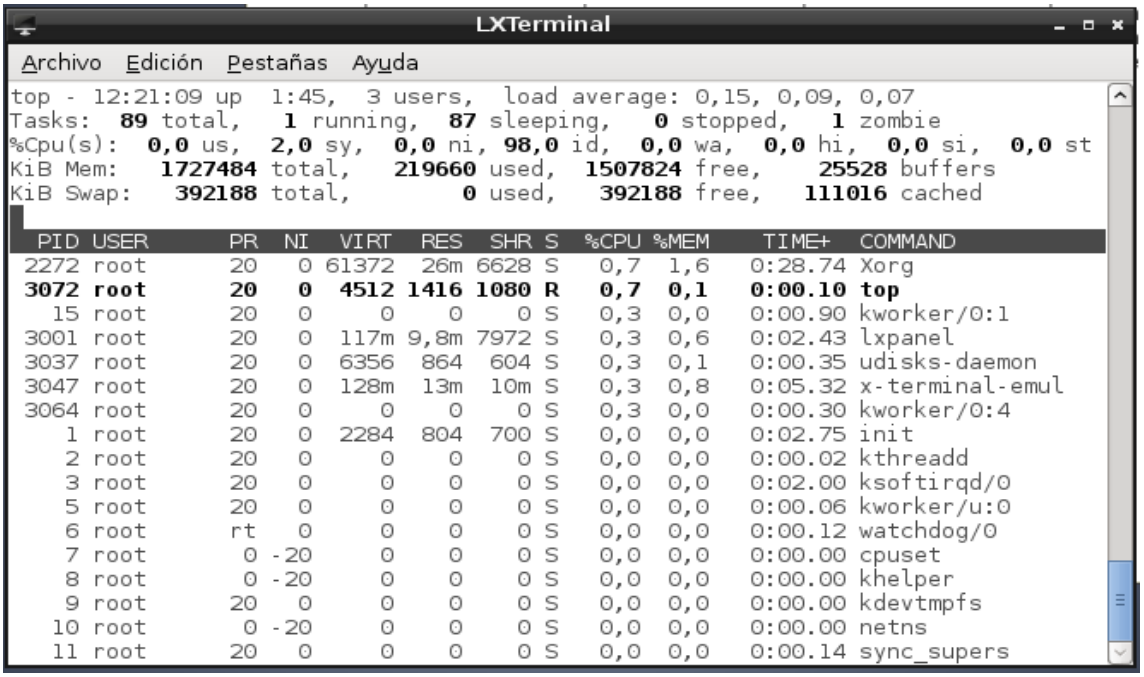
```
F UID  PID     PPID PRI  NI    VSZ   RSS WCHAN  STAT  TTY    TIME COMMAND
4   0  2611     1   20   0    2284   832 -      Ss+   tty1   0:00 /sbin/getty 38400 tty1
4   0  2612     1   20   0   2284    836 -      Ss+   tty2   0:00 /sbin/getty 38400 tty2
4   0  2613     1   20   0   2284    836 -      Ss+   tty3   0:00 /sbin/getty 38400 tty3
4   0  3023  3018   20   0   7320   4448 -      Ss+   pts/0  0:00 -bash
4   0  3502  3497   20   0   7320   4448 -      Ss    pts/1  0:00 -bash
0   0  5171  3502   20   0   4068    856 -      R+    pts/1  0:00 ps l
```

La opción –l nos muestra un listado extendido de los procesos, en esta salida podemos ver el estado de cada proceso, en la columna STAT.

top	muestra información en tiempo real referida a los procesos.

Sintaxis:

```
top
```



Refresca la información cada n segundos. Podrá salir del comando con la tecla q.

En la primera línea nos muestra:

- Hora actual.
- Tiempo que ha estado el sistema encendido.
- Cantidad de usuarios.
- Carga media en intervalos de 5, 10 y 15 minutos respectivamente.

La segunda línea muestra el total de tareas y procesos, los cuales pueden estar en diferentes estados.

En la tercera línea nos muestra los porcentajes de uso del procesador diferenciado por el uso dado:

- **us (usuario)**: tiempo de CPU de usuario.
- **sy (sistema)**: tiempo de CPU del kernel.
- **id (inactivo)**: tiempo de CPU en procesos inactivos.
- **wa (en espera)**: tiempo de CPU en procesos en espera.
- **hi (interrupciones de hardware)**: interrupciones de hardware.
- **si (interrupciones de software)**: tiempo de CPU en interrupciones de software.

En la cuarta línea muestra el uso de la memoria.

- Memoria total.
- Memoria utilizada.
- Memoria libre.
- Memoria utilizada por buffer.

En la quinta línea muestra el uso de la memoria virtual.

- Memoria total.
- Memoria usada.
- Memoria libre.
- Memoria en caché.

Luego en las columnas muestra la siguiente información:

PID: es el identificador de proceso. Cada proceso tiene un identificador único.
USER: usuario propietario del proceso.
PR: prioridad del proceso. Si en esta columna figura *RT* es que el proceso se está ejecutando en tiempo real.
NI: asigna la prioridad. Si tiene un valor bajo (hasta -20) quiere decir que tiene más prioridad que otro con valor alto (hasta 19).
VIRT: cantidad de memoria virtual utilizada por el proceso.
RES: cantidad de memoria RAM física que utiliza el proceso.
SHR: memoria compartida.
S (ESTADO): estado del proceso.
%CPU: porcentaje de CPU utilizado desde la última actualización.
%MEM: porcentaje de memoria física utilizada por el proceso desde la última actualización.
TIME+: tiempo **total** de CPU que ha usado el proceso desde su inicio.
COMMAND: comando utilizado para iniciar el proceso.

Creación de procesos

Cuando un usuario introduce un comando, el shell tras analizar la línea de comando, decide si se trata de un comando propio del shell o bien un comando externo que reside en disco.

En el primer caso, el proceso se lleva a cabo mediante llamadas al sistema, que consisten en peticiones a los servicios que proporciona el núcleo, siendo la única forma que tiene el proceso de acceder al hardware del sistema.

Si se trata de la segunda posibilidad, el shell emite una llamada a *fork( ), que* provoca que el kernel del sistema operativo cree un clon del proceso que realizó la llamada (proceso padre), y hace correr independiente al nuevo proceso creado (proceso hijo), este buscará la trayectoria donde localizar el comando invocado almacenando el entorno del proceso padre y preparando los archivos de entrada y salida estándar. Si el comando es localizado, el shell invocará la llamada *exec( )* para que ejecute el nuevo programa.

De todos modos, el shell será para cada usuario, el padre de los procesos que él ejecute. A partir del shell se crea la estructura jerárquica de los procesos que se ejecutan desde cada terminal. En otras palabras, los procesos que se ejecutan en cada terminal, son hijos del shell asociado con dicha terminal. Si eliminamos al shell se eliminan todos los procesos hijos, salvo que algún proceso se "independice" del proceso padre.

Todo proceso proviene de otro, denominado proceso padre. Solamente existe un proceso que no tiene padre, el proceso *init* con PID 1.

El subdirectorio `/proc/` - Un sistema de archivos virtual

En Linux, todo se guarda en archivos. La mayoría de los usuarios están familiarizados con los dos primeros tipos de archivos, de texto y binarios. Sin embargo, el directorio /proc contiene archivos que no son parte de ningún sistema de archivos asociado a los dispositivos físicos de almacenamiento como discos duros, CD-ROM o pendrives (excepto la RAM). Estos archivos forman parte de un *sistema de archivos virtual* habilitado o deshabilitado en el kernel de Linux cuando está compilado.

Los archivos virtuales poseen cualidades únicas. En primer lugar, la mayoría de ellos tienen un tamaño de 0 bytes. Sin embargo, cuando se visualiza el archivo, éste puede contener una gran cantidad de información. Además, la mayoría de configuraciones del tiempo y las fechas reflejan el tiempo y fecha real, lo que es un indicativo de que están siendo constantemente modificados.

Por cada uno de los procesos que se está ejecutando en el sistema, existe en */proc/* un subdirectorio cuyo **nombre es el número del proceso (PID),** que contiene la información sobre el proceso.

Ejemplo

Si el sistema le asigna el PID 1175 al proceso que resulta de la ejecución del comando *find / -name listado*, se crea en /proc el directorio 1175. Podemos ejecutar el comando *ls –l* del directorio correspondiente a ese proceso de usuario, y visualizar los siguientes archivos:

```
$ls -l  /proc/1175
total 0
-r--r--r--    1   usuario1     usuario1        0     jun 5 19:32   cmdline
lr--r--r-     1   usuario1     usuario1        0     jun 5 19:32   cwd -> /usr/include/pgsql
-r--r----     1   usuario1     usuario1        0     jun 5 19:32   environ
lrwxrwxrwx    1   usuario1     usuario1        0     jun 5 19:32   exe ->/usr/bin/find
dr-x------    2   usuario1     usuario1        0     jun 5 19:32   fd
pr—r--r--     1   usuario1     usuario1        0     jun 5 19:32   maps
-rw------     1   usuario1     usuario1        0     jun 5 19:32   mem
lrwx------    1   usuario1     usuario1        0     jun 5 19:32   root ->/
-r--r--r--    1   usuario1     usuario1        0     jun 5 19:32   stat
-r--r--r--    1   usuario1     usuario1        0     jun 5 19:32   statm
-r--r--r--    1   usuario1     usuario1        0     jun 5 19:32   status
```

Los archivos dentro del directorio /proc/ pueden ser visualizados usando los comandos *cat, more, o less* lo que proveerá gran cantidad de información acerca del sistema y los procesos.

Si visualizamos el contenido del archivo */proc/1175/status,* nos mostrará información referida al proceso tal como: el usuario que lo lanzó, el proceso padre, el estado, el tamaño en KB, etc.

Un administrador de sistemas puede utilizar el subdirectorio */proc* como método sencillo de información de acceso sobre el estado del kernel, los atributos de las máquinas, los estados de los procesos individuales y mucho más.

Archivos virtuales tales como */proc/interrupts, /proc/meminfo, /proc/mounts, y /proc/partitions* proveen una vista rápida actualizada del hardware del sistema. Otros, como */proc/filesystems* y el directorio */proc/sys/* proveen información de configuración y de las interfaces. En líneas generales, los archivos que contienen información sobre un tema parecido se agrupan en directorios virtuales.

En general, todos los archivos que se encuentran en el directorio */proc* solamente se pueden leer. Sin embargo, algunos se pueden usar para ajustar la configuración del kernel. Esto ocurre con los archivos del subdirectorio */proc/sys/.*

Piping o Comunicación de los procesos

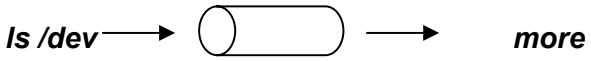
De la misma manera que la salida estándar se puede guardar en un archivo como ya lo vimos en redireccionamiento, también es posible enviar esa salida a un pipe, desde el cual un segundo comando tomará su entrada. Es lo que se llama proceso *piping (proceso de conexión),* porque se crea un pipeline (tubería o conducto), que consiste en un canal de información que recibe datos por un extremo y los transmite (modificados) por el otro. Por un extremo del pipe se escribe y del otro se lee. Los pipes son creados y destruidos por el kernel, en la memoria.

El piping es un mecanismo que permite que los comandos de Linux se combinen fácilmente desde la línea de comandos. Es muy eficiente ya que, la salida del primer programa alimenta directamente al segundo programa, sin tener que pasar por la escritura de archivos de disco temporales.

Las tuberías pueden estar formadas por un número "ilimitado" de comandos. Estos no se ejecutan secuencialmente, o sea, no se espera a que termine uno para ejecutar el siguiente, sino que se va haciendo de forma concurrente. La conexión o piping se indica con la barra vertical de pipe "|" entre los comandos.

Ejemplo 1

```
$ls  /dev   |   more
```



Si se ejecuta el comando *ls* /dev se sabe que es posible que existan más de 25 archivos para listar, algunos nombres desaparecerán de la pantalla antes de que pueda leerlos. Haciendo una tubería al comando *more,* este programa producirá una pausa después de cada página de pantalla de salida y deberá pulsar la tecla Space, cuando desee ir a la página siguiente, o intro para avanzar de a una línea por vez, de esa manera la salida de ls será mostrada en forma paginada.

Ejemplo 2

```
$ls  /bin  | wc -w
```

Muestra cuántos archivos contiene el directorio /bin. El comando *wc –w* cuenta las palabras que se muestran con el comando ls.

Ejemplo 3

```
$who | wc -l
```

Muestra la cantidad de sesiones abiertas. El comando *wc –l* cuenta las líneas generadas por who.

Ejemplo 4

```
$who  | grep  juan

juan     tty02 Sep 15         16:34
```

El comando *who* genera un listado de las sesiones de usuario abiertas, *grep* busca en ese listado la cadena de caracteres *juan* y muestra por pantalla las líneas en donde la encuentra. Nos permite saber si el usuario *juan* ha iniciado su sesión, a qué hora y en cuál terminal.

Procesos Background o en Segundo Plano

La multitarea de Linux le permite ejecutar varios procesos a la vez, pero al lanzar un proceso desde un shell, no podemos volver a lanzar otro proceso desde él, hasta que finalice la ejecución del primero. Sin embargo, podemos hacer que el shell quede disponible para poder lanzar nuevos procesos en el momento de ejecutar un comando, lanzándolo en lo que se conoce como **modo subordinado -background- o ejecución en segundo plano**, lo que permite retomar el control del teclado y pantalla para lanzar nuevos procesos.

Para ello se utiliza el operador **&** al final de la línea de comando de la siguiente forma:

```
comando &
[n° de tarea]  PID
```

Al ejecutar un proceso en modo subordinado, el shell devuelve un número de tarea y un identificador de proceso o PID con el que nos puede referir al trabajo subordinado.

Normalmente las entradas, salidas y salida de error estándar de procesos subordinados, se redirigen para que la sesión de la terminal no se vea interrumpida por su salida, de esta forma:

```
find /etc /proc -name group > archivo  2> /dev/null&
[1] 2471
```

Linux permite tantos trabajos subordinados como queramos, aunque el rendimiento se reducirá. Al finalizar un proceso subordinado no se muestra ninguna notificación. Podemos chequear el estado del proceso con el comando *ps*, y el estado de las tareas con el comando *jobs*.

jobs	muestra las tareas/procesos que se ejecutan en segundo plano y los que están suspendidos.

Mediante el comando *jobs* es posible visualizar todos los procesos en segundo plano que se están ejecutando. Si la usa obtendrá una salida como la siguiente:

Ejemplo

```
$jobs
[1]    ejecutando    find / -name  temp* > listatemp&
[2]-   ejecutando    ls -lR /  > dirlist&
[3]+   detenido      sleep 400
```

En este ejemplo, el último trabajo se marca con un más + y el penúltimo con un menos (–) el resto de los trabajos no llevan marcas.

nohup	permite que el comando se ejecute en modo subordinado y se ejecute hasta finalizar, aún después que la sesión haya terminado.

Los procesos subordinados son eliminados al terminar la sesión de trabajo, ya que están asociados al shell de presentación. Pero podemos hacer que los procesos subordinados sigan ejecutándose después de cerrar la sesión de trabajo, mediante el comando *nohup*.

Ejemplo

```
$nohup find / -name  temp* > listatemp&
```

Este comando hace que el comando se ejecute en modo subordinado y continúe hasta su fin.

Si el comando produce salidas a la terminal, se almacenarán en este caso en un archivo *nohup.out,* a menos que se redirijan esas salidas (en este caso *listatemp*), ya que al haber cerrado la sesión no existe terminal en donde visualizarlas.

Detención y relanzamiento

Como vimos, podemos detener temporalmente un proceso con el comando *Ctrl.+z*, de esa forma se retoma el control del shell para ejecutar una segunda tarea.

Ejemplo

Lanzamos el comando *find* en primer plano y luego pulsamos Ctrl+z, el trabajo se detiene temporalmente. El mismo resultado obtenemos si enviamos al proceso la señal SIGSTOP.

```
$find / -name temp* > listatemp
Ctrl+z

[1] + Stopped   find / -name temp* > listatemp
```

fg	Reanuda trabajos suspendidos poniéndolos en *foreground* (primer plano) o trabajos en *background* los pasa a primer plano.

Sintaxis:

```
fg  [nro.de tarea]
```

En el ejemplo anterior, lanzamos en primer plano el comando *find*, luego lo suspendimos. Con el comando:

```
fg  1
```

La tarea 1, (`find / -name temp* > listatemp`) continua su ejecución en primer plano.

También podemos usar el prefijo **%** para indicar el nº de tarea o el nombre del comando. Este formato es utilizado también por el comando *bg.*

`fg %1`	pone en primer plano la tarea 1.
`fg %find`	pone en primer plano la tarea cuyo nombre comienza con *find.*
`fg %?name`	pone en primer plano la tarea que contenga la cadena *name*.
`fg %%`	pone en primer plano la última tarea ejecutada.

Ejemplo

```
$top > controlprocesos&
[1]  3221

fg  1
```

La tarea 1 continúa su ejecución en primer plano.

bg permite que un trabajo continúe su ejecución en segundo plano.

Sintaxis:

```
bg  [nro.de tarea]
```

Si sólo hay un trabajo parado o en la lista de jobs no es necesario poner el número de tarea.

Ejemplo

`$sleep 300` lanza en primer plano el proceso sleep (duerme por 300 segundos)
`Ctrl+z` suspende el proceso

```
[1] 3445
```

`$bg  1` reanuda la ejecución pero en segundo plano.

kill permite enviar una señal a uno o más procesos que se estén ejecutando en el sistema.

Sintaxis:

```
kill –señal proceso
```

La señal puede ser indicada con un número/palabra. Es posible indicar el proceso al cual se envía la señal por su **PID** o su nombre. Al escribir Ctrl+c en el shell donde se ejecuta un proceso el sistema le envía una señal SIGINT, que por defecto causa la terminación del proceso. Ctrl+z hace que el sistema envíe una señal SIGTSTOP que suspende la ejecución del proceso.

Con este comando podemos eliminar un proceso que, por ejemplo, esté consumiendo mucho tiempo de CPU, no produzca la salida esperada, produzca una salida excesiva, tenga error de programación, esté en estado bloqueado, o simplemente sea un proceso muy largo. Un usuario puede eliminar cualquier proceso del cual sea propietario y el superusuario, cualquier proceso del sistema, excepto los pids 0, 1, 2, 3 y 4.

Si el proceso a eliminar está ejecutándose en modo interactivo, se hará pulsando Ctrl+c ó Ctrl+d (según el shell utilizado). Sin embargo, para un proceso subordinado no existe terminal asociado, por lo que no podremos eliminarlo de esa manera. Para hacerlo, utilizamos el comando **kill** con el PID del proceso o el identificador de tarea (con el símbolo % delante) que se desea eliminar como argumento. Tras ejecutar este comando, el proceso desaparecerá de la tabla de procesos. Se pueden especificar múltiples PIDs como argumentos de la orden *kill* separados por espacios.

```
kill PID1 PID2 PID3
```

Opción:

-l lista el nombre de las señales.

```
root@debian-monica:~# kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Podemos especificar en el comando kill el nombre o el número de la señal.

Ejemplos

```
$kill  3487  4012
```

Termina la ejecución de los procesos de pid nº 3487 y 4012. La señal TERM (nº 15), que es la señal por defecto si no se especifica otra. Así que para solicitar el fin de ejecución de uno o varios procesos únicamente tenemos que ejecutar kill seguido de los PID correspondientes.

```
$kill -9 4012
```

La señal SIGKILL (nº 9) fuerza a los procesos a terminar inmediatamente sin permitirles terminar de forma limpia, es decir, los procesos no terminan las peticiones pendientes.

```
$kill -SIGSTOP  4012    o        $kill -19 4012
```

La señal SIGSTOP (nº19) suspende el proceso.

```
$kill -SIGCONT   4012    o    $kill  -18  4012
```

La señal SIGCONT (nº18) reanuda la ejecución del proceso suspendido.

```
$kill %1
```
mata la tarea nº 1.

Planificación de procesos

Linux utiliza un algoritmo simple de planificación[12] basado en prioridades dinámicas, es decir que la prioridad se ajusta de acuerdo al tiempo que los procesos han esperado y al tiempo que han usado el CPU. Cuenta con dos tipos de prioridades: normal y real-time. A cada proceso se le permite ser ejecutado por un breve período de tiempo por ejemplo de 200ms (time-slice ó quantum).

Los procesos de Linux son *preemptive,* pues el planificador expropiativo asigna un tiempo límite después del cual el proceso actual es suspendido, para dar paso a otro proceso y mantener así la ilusión de concurrencia. Sin embargo el kernel es *non-preemptive,* es decir, una vez que se

[12] En POSIX, cada política de planificación tiene asociada al menos 32 niveles de prioridad, el planificador elige el proceso con la mayor prioridad. En Solaris, el manejo de los procesos es por prioridad y round robin. En algunas versiones se maneja también un ajuste dinámico de la prioridad.

asigna CPU a un proceso, dicho proceso se apropia del CPU hasta que termine, ó por sí mismo ceda el CPU a otro proceso.

Además, Linux tiene herramientas para temporización y planificación de tareas, pero el usuario tiene cierta capacidad de modificar la planificación que el Sistema Operativo hace, modificando la prioridad de los procesos, o haciendo que los procesos que no precisan rapidez en su ejecución, se ejecuten en determinados intervalos de tiempo en los que el sistema está más desocupado. Para esto último, Linux nos ofrece varios comandos: nice, renice, at, batch y la utilidad cron.

Prioridades de los procesos - nice y renice

Normalmente, a todos los procesos de tiempo compartido se les asigna la misma cuota de tiempo, por lo que se dice que tienen la misma prioridad, pero Linux dispone de herramientas para modificar la prioridad de un proceso. El incremento de la prioridad de un proceso, generalmente, hace que se ejecute más rápidamente a costa de que los otros procesos deban esperar. Al contrario, al reducir la prioridad, un proceso demanda menos tiempo de CPU durante cada intervalo, tardando así más en completar su ejecución.

Valor de la prioridad PRI:

- Mayor prioridad: el valor de PRI es cercano a 1.
- Menor prioridad: el valor de PRI es cercano a 100.

El kernel es quien asigna las prioridades iniciales, asignándole a los procesos del sistema como por ejemplo manejo de disco, terminales, etc; prioridades más altas para que sean atendidos antes que los procesos de usuarios normales, y a los procesos que realizan muchos cálculos, les otorga prioridades bajas de manera que el uso de la CPU sea equilibrado.

De todos los procesos que están en RAM listos para ejecutarse, recibe el uso de la CPU el que tenga mayor preferencia, haciendo uso de la prioridad asignada a cada uno, la cual se calcula dinámicamente a partir de la prioridad base y los valores C (tiempo utilizado/tiempo asignado de CPU) y NI (nice).

Sólo el superusuario tiene privilegios para incrementar la prioridad a un proceso, pero cada usuario puede decrementar la prioridad de sus procesos mediante el comando nice.

nice permite asignar una determinada prioridad a un proceso que está por comenzar su ejecución.

Sintaxis:

```
nice –NI comando
```

El kernel puede asignar una prioridad base de por ejemplo 80 para un proceso en particular, pero el rango del incremento a utilizar para los comandos nice y renice es de –20 a 20, si usamos un valor negativo aumentamos la prioridad del proceso, por el contrario si usamos un valor positivo la prioridad disminuye. Si no indicamos el valor de NI, toma por defecto el valor 10.

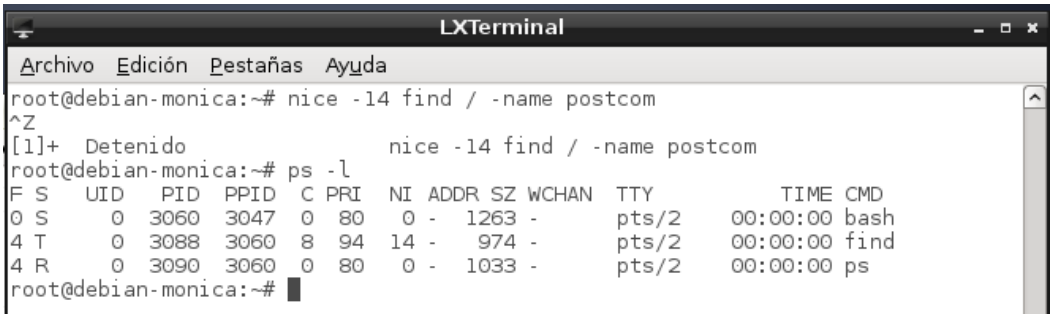
-NI suma el valor a PRI
--NI resta el valor a PRI

Ejemplos

```
$nice find  / -name postcom  o nice 10 find  / -name postcom
```

En este caso el comando *nice* incrementa el valor absoluto de la prioridad en 10 unidades (por defecto) por lo que el proceso tiene menor prioridad.

```
$nice  -14 find  / -name postcom
```



En este caso *nice* hace que el proceso *find* incremente en 14 unidades el valor absoluto de su prioridad por lo que el proceso tiene **menor prioridad**. Esto lo podemos ver con el comando *ps –l*, en las columnas PRI y NI.

```
$nice  - - 8 find  / -name postcom
```

Decrementa en 8 unidades el valor de la prioridad del proceso *find,* lo que implica que el proceso tiene **mayor prioridad**.

renice	permite modificar la prioridad a un proceso que se encuentra en ejecución.

Cada usuario sólo podrá modificar sus propios procesos. El superusuario (root), sin embargo, podrá modificar sus procesos y los de cualquier otro usuario.

Sintaxis:

```
renice -/+NI n° de proceso
```

+NI suma a PRI por lo que el proceso tendrá menor prioridad.
-NI resta a PRI por lo que el proceso tendrá mayor prioridad.

Ejemplos

```
$renice +10  3113   o    renice 10  3113
```

En este caso el proceso cuyo PID es 3113 disminuirá su prioridad, el valor de PRI aumentó en 10.

```
$renice -15  3113
```

En este caso, el proceso 3113 queda con mayor prioridad porque el valor de PRI disminuyó en 15 unidades.

Ejecutar tareas en forma programada - at, batch, cron

Linux ofrece la capacidad de programar procesos para que se ejecuten en un momento determinado, incluso sin necesidad de iniciar la sesión de usuario.

at	permite planificar trabajos para una única ejecución posterior.

Utilice *at* cuando quiera ejecutar un comando o trabajo en una fecha y hora determinadas, el trabajo se ejecutará siempre y cuando, el sistema esté en marcha. El comando *at* leerá de la entrada estándar el/los comandos a ejecutar en esa hora, fecha indicada en *time*, finalizando la carga con Ctrl+d.

Sintaxis:

```
at time
          >orden 1
          >orden 2
          >orden 3....
          Ctrl+d
```

Este comando ejecutará las órdenes a la hora especificada del día actual. El comando nos devuelve, el número de trabajo y la hora en la que se ejecutará. La salida de la orden, puesta en cola, se enviará al usuario, como correo, a menos que se redireccione la salida de una o más ordenes en particular, a un archivo específico.

Cuando se especifica *time* se pueden utilizar varias opciones:

- HH:MM hora y minutos. Se puede añadir el sufijo am o pm.
- midnight (24:00 o 12:00pm), noon (12:00 o 12:00am) y teatime (16:00)
- MMDDAA , MM/DD/AA , DD.MM.AA, se pueden utilizar para indicar fechas como 022517
- now: especifica la hora actual
- tomorrow: el día siguiente
- Si se usa el signo + se puede especificar desplazamientos en minutos, horas, días, semanas o meses (minutes, hours, days, weeks, months). Por ejemplo: noon + 2 days. Por ejemplo:

Ejemplos

at 19	**#** a las 7:00 de la tarde
at 6:30am	**#** a las 6:30 de la mañana
at now	**#** ahora
at midnight	**#** a medianoche
at noon + 15 minutes	**#** 15 minutos después del mediodía
at 4:10pm + 1 months	**#** a las 4:10 dentro de 1 meses
at 13:50pm Jul 16, 2019	**#** a las 13:50 hs del 16 de julio del año 2019

batch	se usa para programar que se ejecute una única tarea cuando la carga promedio de los sistemas estén por debajo de 0.8

Este comando le dice al shell que ejecute la tarea ahora, pero en realidad los trabajos serán atendidos tan pronto como la carga del sistema sea suficientemente baja, es decir cuando el sistema esté menos ocupado. Esto es útil cuando se desean ejecutar muchos trabajos largos de una vez. Con lo que se evita que todos los procesos intenten ejecutarse a la vez y acaparen el CPU.

Sintaxis:

```
batch –opciones time
```

Opciones:

-f archivo lee los comandos que se van a planificar desde el archivo que se especifica, en lugar de pedirlos al usuario.

-m una vez que ha terminado un trabajo, manda un mensaje al usuario que lo planificó e incluye en el cuerpo del mensaje cualquier salida que haya generado.

Cuando se especifica tiempo se pueden utilizar varias opciones:

- HH:MM hora y minutos. Se puede añadir el sufijo am o pm.
- midnight (24:00 o 12:00pm), noon (12:00 o 12:00am) y teatime (16:00)
- MMDDAA , MM/DD/AA , DD.MM.AA, se pueden utilizar para indicar fechas como 022517
- now especifica la hora actual
- tomorrow el siguiente día
- si se usa el signo + se puede especificar desplazamientos en minutos, horas, días, semanas o meses (minutes, hours, days, weeks, months). Por ejemplo: noon + 2 days.

atq	muestra los trabajos pendientes de ejecutar por el comando at.

Muestra los trabajos programados por el usuario, siendo root muestra todos los trabajos programados.

Sintaxis:

```
atq [opciones]
```

cron

El proceso *cron* es un demonio que se ejecuta al iniciar el sistema y se activa cada minuto y comprueba si hay trabajos programados para ese minuto. El demonio cron se inicia de */etc/rc.d* o */etc/init.d* dependiendo de la distribución.

Una vez que empieza a ejecutarse *cron*, cada minuto chequea la cola de trabajos planificados con *at*, así como los archivos *crontab* para ver si debe ejecutar algún proceso, lanzándolo en el momento adecuado. Si no hay nada que hacer, pasa a estado Durmiendo, permaneciendo inactivo hasta el próximo chequeo en el siguiente minuto.

A diferencia de las órdenes *at y batch* que sirven solo para planificar tareas una única vez. Linux mantiene un conjunto de tablas cronológicas que contienen un conjunto de tareas planificadas para que se ejecuten periódicamente. Existe una tabla o archivo por cada usuario que ha enviado un trabajo y el sistema ejecuta ese trabajo utilizando la cuenta del usuario.

Programar trabajos es una tarea fácil, cada usuario crea un archivo de texto con cualquier nombre, que contiene una línea por cada trabajo planificado (conocido como archivo crontab). Pero, cada usuario puede tener su propio archivo *crontab* que por defecto se almacena en el directorio */var/spool/cron/contabs*.

crontab	se utiliza para programar tareas, es decir, programar la ejecución de otros comandos.

Sintaxis:

```
crontab [opciones] file
```

En el archivo *file* se especifica en cada línea, el tiempo en que queremos ejecutar un comando (puede contener tantas líneas como queramos), especificando ese tiempo en términos de:

minutos (0-59)
horas (0-23)
días del mes (1-31)
mes del año (1-12)
días de la semana (0-6:domingo a sábado). O bien: (1-7: lunes a domingo)

El formato de las líneas de este archivo será:

```
minuto hora díadelmes mesdelaño díadelasemana  comando
```

Ejemplo

Podemos crear el archivo tareas1 con un editor de texto.

```
$cat tareas1

30 20 30 * * tar cvf ~/respaldo.tar ~/archivos/*.dat| gzip –9 ~/respaldo.tar
```

Luego programarla como:

```
$crontab tareas1
```

Debemos especificar todos los campos, pudiendo colocar un * en cualquiera de ellos para indicar que se ejecute la orden en cualquier momento válido para ese campo. También podemos especificar una secuencia de valores en un campo cualquiera usando comas para separar cada valor. La orden puede contener pipes, redireccionamientos y cualquier aspecto válido en un shell.

Tanto *at, batch* como *crontab* proporcionan mecanismos de seguridad que pueden impedir a los usuarios planificar trabajos sin autorización. Las listas de usuarios autorizados y no autorizados se encuentran en el directorio */etc/cron.d* en los archivos *at.allow*, *at.deny, cron.allow*, cron.*deny* respectivamente, con una línea por usuario. Así por ejemplo, si el archivo *at.allow* no existe, los usuarios que no pueden ejecutar *at* estarán indicados en *at.deny*. Si ambos archivos no existen, sólo el usuario root puede programar con *at* o *batch*. Sólo el superusuario puede modificar estos archivos.

uptime	muestra un informe sobre el desempeño del sistema.

Todo administrador de sistemas debe contar con toda la información disponible del funcionamiento del sistema para poder realizar cualquier planificación de tareas, *uptime* es un comando que proporciona un informe breve sobre el desempeño del sistema.

Este comando visualiza la hora actual, la cantidad de tiempo que el sistema lleva funcionando, es decir, el tiempo transcurrido desde la última vez que se inició el sistema, la cantidad de usuarios que están actualmente en el sistema. Además visualiza el número medio de trabajos esperando a ejecutarse desde el último, los cinco y los diez últimos minutos. Si ese número se acerca a 0 significará que el sistema está bastante desocupado, mientras que un valor cercano a 1 indica que el sistema está bastante cargado.

Ejemplo

```
$uptime
  8:00pm up 142 days, 6:25, 2 users, load average: 0.51,  0.52,  0.52
```

Actividad 1

1. ¿Para qué se utiliza el PID de un proceso?

2. ¿Pueden existir dos PID iguales en un instante dado? ¿Por qué?

3. ¿Puede un usuario modificar la prioridad de un proceso perteneciente a otro usuario? ¿Por qué? ¿En qué casos podría realizarlo?

4. ¿Qué interpretación le da Ud. al caracter '**?**' de la columna TTY en la salida del comando ps?

5. Investigue qué información podemos visualizar sobre el proceso 1.

6. ¿Cuál es el tamaño (size) de los archivos del subdirectorio /proc? ¿Por qué?

Actividad 2

1. Iniciar su sesión como usuario común.

```
Login
```

2. Listar sus procesos activos.

```
ps
```

3. Realizar un informe de TODOS los procesos activos en el sistema. Analice su salida.

```
ps –aux  | more
```

4. Si la salida del comando ps es :

PID	TTY	STAT	TIME	COMMAND
8832	p1	R	0:00	ps
30674	p1	S	0:00	/bin/login –h
29645	p1	T	0:00	analog var/log/httpd/access_log

5. Eliminar el proceso access_log. Verificar la eliminación.

```
kill  -9  29645
ps
   PID   TTY   STAT  TIME  COMMAND
   8832    p1     R  0:00  ps
   30674   p1     S  0:00  /bin/login –h
```

6. Ejecutar el comando *uptime* y comentar su resultado.

7. Realizar la búsqueda del archivo at.allow en todo el sistema de archivos

```
find / –name at.allow
```

8. Detener el comando anterior y enviarlo a 2° plano.

```
Ctrl+z# devuelve el número de trabajo por ejemplo:
[3] Stopped            find  / -name at.allow

   Luego:
   bg 3
```

9. Volver a poner en primer plano el proceso de búsqueda anterior.

```
fg 3
```

10. Usted es el Administrador del Sistema, y a las 2000 horas comenzarán tareas de mantenimiento del Sistema, se pide programar 20 minutos antes, un listado de los procesos activos en ese momento y un listado de los usuarios conectados al Sistema:

```
at 19:400 (enter)
        >ps –a (enter)
        >who (enter)
         Ctrl+D
```

11. Mostrar por pantalla las tareas planificadas en el punto anterior.

```
at –l
```

12. Verificar la salida de punto 10.

```
mail

&Nro de mail
```

13. Buscar el texto “cron” en todos los archivos de su directorio de login cuya extensión sean **txt** y crear un archivo *salidacron* con el resultado de la búsqueda. Realizar esta tarea en segundo plano y cuando el sistema esté menos ocupado o la carga del mismo así lo permita (batch).

```
batch
grep  cron ~/*.txt  >  salidacron&
Ctrl+D
```

14. Verificar la realización del comando anterior.

```
more salidacron
```

Actividad 3

1. Analizar los números de PID del /proc.

2. Hacer un listado recursivo y paginado de su directorio de login.

```
ls –R | more
```

3. Buscar los archivos cuyo nombre comience con lin, en todo el file system y redireccionar la salida al archivo *listalin*. Ejecutar en segundo plano.

```
find / –name lin* > listalin&
```

4. Verificar en qué estado se encuentra el proceso anterior. ¿Está en ejecución? ¿Qué PID tiene asignado?

```
jobs
```

5. Determinar si el proceso 1456 se está ejecutando en:

a) Su sesión de trabajo: `ps | grep 1456`
b) En todo el sistema: `ps –aux | grep 1456`

6. Lanzar en background un proceso que clasifique por número de i-nodo en orden ascendente los archivos.

```
ls –i  ~/  |  sort –n&
```

7. Usted es un encargado de área de una organización, ha ingresado al sistema como **usuario10** y a las 20:00 horas termina su jornada de trabajo. Entonces decide programar 30 minutos después:
   a) Realizar una copia de los archivos que hay en su directorio de trabajo a un directorio de respaldo /home/"nombre_usuario"/respaldo
   b) Conocer los usuarios que estén conectados al Sistema en ese momento.

```
at 20:30
cp  * .* /home/usuario10/respaldo
who
ctrl+d
```

8. Mostrar por pantalla las tareas planificadas en el punto anterior.

```
at -l
```

9. Buscar en el File System el archivo *at.deny***.** Realizar esta tarea en segundo plano y cuando la carga del sistema así lo permita (modo batch).

```
batch
find  /  -name at.deny&
ctrl+d
```

10. Verificar la realización del comando anterior. ¿Qué comando utilizó?

```
mail
```

11. A causa de un factor externo, la tarea de copia del ejercicio 7-a se debe completar en forma urgente. Usted es supervisor. Asigarle la máxima prioridad de ejecución y tener en cuenta que el proceso ya se está ejecutando. ¿Qué pasos realizará?

```
ps -l
renice  -20  [pid obtenido]
ps -l
```

Actividad 4

1. Generar un archivo con información sobre los usuarios conectados al sistema pero no se incluya usted en ese informe.

2. Programar la siguiente tarea: generar un archivo con un informe sobre el desempeño del sistema cada 60 minutos (tiempo en el que el sistema lleva funcionando, usuarios conectados, etc).

3. Realizar un informe de todos los procesos activos del usuario *alumno90* y analizar su salida.

4. Indicar cuál es la función comando del renice.

Capítulo 9

Administración de Memoria

Administración de Memoria en Linux

La Memoria Principal es un recurso valioso que se debe asignar y compartir entre los procesos activos. Mantener en memoria la mayor cantidad posible de procesos, permite hacer un uso eficiente del procesador y de los servicios de E/S. El micro no tendrá tiempos ociosos pues siempre habrá un proceso para ejecutar. Por ello, una de las tareas más importantes y complejas de un sistema operativo es la gestión de la memoria. El sistema operativo Linux utiliza memoria virtual y para su administración, las técnicas de *intercambio y paginación por demanda*. Para ello, se vale de las siguientes estructuras de datos:

- **Tabla de páginas**: describe las páginas virtuales del proceso.
- **Descriptor de bloques de disco**: describe el bloque de disco que contiene una página determinada.
- **Tabla de marcos de páginas**: describe cada marco de página de la memoria real.
- **Tabla de intercambios**: registra las páginas que están en cada uno de los dispositivos de intercambio, pues podemos definir más de uno.

La memoria física se divide en *marcos* de página y la memoria virtual se divide en *páginas*. La asignación de una página de memoria virtual a un marco físico se registra en la *tabla de páginas*. La unidad de gestión de memoria *MMU* realiza una traducción de la dirección de memoria virtual a la dirección de memoria física y verifica si la página a la cual quiere acceder está cargada o no en memoria. Las búsquedas en las tablas de páginas son costosas para el sistema, por lo que algunas arquitecturas[13] mantienen una caché de búsqueda rápida llamada Translation Lookaside Buffer TLB[14] (*buffer de traducción anticipada*). Cualquier dirección de memoria virtual que requiera traducción a la dirección de memoria física, se compara primero con el buffer de traducción para una asignación válida. Cuando una traducción de dirección válida no está presente en el TLB, se denomina “fallos de TLB”. En ese caso la unidad de gestión de memoria tendrá que consultar la tabla de páginas, y una vez encontrada la página, deberá realizar el intercambio de páginas entre la memoria y el disco. Dada la carga que significa para el sistema las búsquedas realizadas sobre las tablas de página, éste debe procurar reducir los fallos de TLB.

Con la característica HugeTLB en el kernel de Linux, las aplicaciones pueden beneficiarse con la asignación de páginas grandes, especialmente las que demandan mucha memoria tales como aplicaciones de bases de datos, aplicaciones HPC[15], etc. Esto permite que la cantidad de fallos de TLB se reduzcan y por lo tanto el costo que generan las búsquedas realizadas sobre las tablas de páginas. La información sobre Hugepages se puede consultar en el archivo */proc/meminfo*.

Ejemplo

```
cat  /proc/meminfo  | grep  Huge
HugePages_Total:4
HugePages_Free:2
HugePages_Rsvd:0
Hugepagesize: 4096 KB
```

En el ejemplo anterior se muestra que están definidas 4 páginas grandes de 4MB c/u, de las cuales 2 han sido asignadas y 2 están libres. El manejo de páginas grandes aún no es totalmente transparente al usuario, por lo que el administrador deberá definir el conjunto de páginas grandes a utilizar y las aplicaciones a las que serán asignadas.

[13] X86 de 32bits tiene soporte para páginas de 2MB y 4MB.
[14] El **TLB** es una tabla asociativa (un *hash*) en memoria de alta velocidad, una suerte de registros residentes dentro de la **mmu**, donde las *llaves* son las páginas y los *valores* son los marcos correspondientes. De este modo, las búsquedas se efectúan en tiempo constante.
[15] High Performance Computing (HPC) aplicaciones que requieren de gran capacidad computacional y grandes cantidades de memoria.

Monitoreando la memoria

Uno de los comandos más utilizados para poder brindar información sobre el uso de la memoria RAM en Linux es:

free	muestra la cantidad total de memoria física y de intercambio presente en el sistema, así como la memoria compartida y los buffers utilizados por el kernel.

Sintaxis:

```
free [-opciones] [-s demora] [-t]
```

Por omisión muestra la cantidad de memoria en Kbytes, pero lo puede hacer en bytes(-b) , en Mbytes (-m), Gigabytes (-g) . Los datos que muestra el comando free los obtiene del archivo /proc/meminfo.

Opciones:

-b	**--bytes**	muestra los valores en bytes.
-k	**--Kilo**	muestra los valores en KBytes.
-m	**--mega**	muestra los valores en MBytes.
-g	**--giga**	muestra los valores en Gbytes.
	--tera	muestra los valores en TBytes.
-h	**--human**	muestra los valores con un máximo de tres dígitos y le agrega la unidad en el que esta expresado. B (bytes), K (kilos), M (megas), G (gigas), T (teras).
-s	**--seconds**	muestra la salida del comando cada n segundos.
-t	**--total**	muestra valores totales de cada columna.

Ejemplo

```
$free -m
             total       used       free     shared    buffers     cached
Mem:          1007        924         82        318         57        673
-/+ buffers/cache:        193        814
Swap:          382          1        381
```

Muestra un informe de la memoria en MBytes.

```
$free  -s 30 -t
             total       used       free     shared    buffers     cached
Mem:       1031828     947376      84452     326260      59508     690584
-/+ buffers/cache:     197284     834544
Swap:       392188       1856     390332
Total:     1424016     949232     474784

             total       used       free     shared    buffers     cached
Mem:       1031828     947476      84352     326260      59516     690584
-/+ buffers/cache:     197376     834452
Swap:       392188       1856     390332
Total:     1424016     949332     474684
```

En este ejemplo, cada 30 segundos el sistema emite un informe con cifras dadas en Kbytes, muestra al final una línea de totales (-t). Puede cortar la salida del comando con las teclas *ctrl+c*.

Archivo de información de memoria

/proc/meminfo brinda información sobre la memoria.

En el directorio /proc, el sistema guarda información referente al hardware del sistema y su configuración, e información sobre los procesos. El sistema de archivos bajo /proc, es un sistema virtual.

El archivo */proc/meminfo* brinda información sobre la memoria, y es utilizado por varios comandos. En las primeras líneas del archivo muestra la memoria total (MemTotal), memoria libre (MemFree), del buffers (Buffers), de la cached (Cached), más adelante el tamaño de la memoria Swap total (swapTotal) y la cantidad de swap libre (SwapFree), luego la cantidad de memoria asignada a la tabla de páginas (PageTables). En la siguiente tabla se detallan otros campos.

Ejemplo

```
more  /proc/meminfo
```

```
redes@simred:~$ more /proc/meminfo
MemTotal:        1031828 kB
MemFree:           79948 kB
MemAvailable:     400296 kB
Buffers:           61316 kB
Cached:           693008 kB
SwapCached:          896 kB
Active:           474400 kB
Inactive:         433916 kB
Active(anon):     263572 kB
Inactive(anon):   216740 kB
Active(file):     210828 kB
Inactive(file):   217176 kB
Unevictable:           0 kB
Mlocked:               0 kB
HighTotal:        141256 kB
HighFree:          15424 kB
LowTotal:         890572 kB
LowFree:           64524 kB
SwapTotal:        392188 kB
SwapFree:         390332 kB
Dirty:               452 kB
Writeback:             0 kB
AnonPages:        153092 kB
CommitLimit:      908100 kB
Committed_AS:    1137084 kB
VmallocTotal:     122880 kB
VmallocUsed:       22424 kB
VmallocChunk:      98748 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       34808 kB
DirectMap2M:      872448 kB
redes@simred:~$
```

Según sea la distribución el archivo tendrá algunos de los siguientes campos:

MemTotal	Cantidad total de RAM física en kilobytes.
MemFree	Cantidad de RAM física, en kilobytes, sin utilizar por el sistema.
Buffers	Cantidad de RAM física, en kilobytes, usada para los archivos de buffers.
Cached	Cantidad de RAM física en kilobytes usada como memoria caché
SwapCached	Cantidad total de swap cache, en kilobytes
Active	Cantidad total de buffer o memoria caché de página, en kilobytes, que está en uso activo
Inactive	Cantidad total de buffer o memoria caché de página, en kilobytes, que está en uso inactivo
HighTotal y HighFree	Cantidad total de memoria libre, que no está mapeada en el espacio del kernel. El valor HighTotal puede variar dependiendo del tipo de kernel utilizado.

LowTotal y LowFree	Cantidad total de memoria libre implantada directamente en el espacio del kernel. El valor LowTotal depende del tipo de kernel utilizado.
SwapTotal	Cantidad total de swap disponible, en kilobytes.
SwapFree	Cantidad total de swap libre, en kilobytes.
Dirty	Cantidad total de buffer y páginas de la caché, en kilobytes, que podrían quedar libres.
Committed_AS	Cantidad total de memoria, en kilobytes, estimadas para completar la carga de trabajo.
PageTables	Cantidad total de memoria, en kilobytes, dedicada al nivel más bajo de la tabla de páginas.
VMallocTotal	Cantidad total memoria asignada, en kilobytes, del espacio total de direcciones virtuales.
VMallocUsed	La cantidad total de memoria utilizada, en kilobytes, de espacio de direcciones virtuales.
VMallocChunk	El bloque continuo de memoria más grande, de espacio de direcciones virtuales disponibles. Expresado en kilobytes.
HugePages_Total	El número total de páginas grandes para el sistema.
HugePages_Free	El número total de páginas grandes disponibles para el sistema. Esta estadística sólo aparece en las arquitecturas x86, Itanium y AMD64.
Hugepagesize	El tamaño para cada unidad de hugepages en kilobytes. Por defecto, el valor es 4096 KB en los kernels de un sólo procesador para las arquitecturas de 32 bits. Para los kernels SMP, hugemem y AMD64, el valor por defecto es 2048 KB. Para las arquitecturas Itanium, el valor por defecto es 262144 KB. Esta estadística solamente aparece en las arquitecturas x86, Itanium y AMD64.

vmstat muestra informes estadísticos sobre el uso de la memoria virtual. Da información sobre procesos, memoria, paginación, E/S y actividades de la CPU.

El primer informe solicitado, da valores medios desde el último arranque del sistema. Los informes adicionales, muestran valores obtenidos en un período de muestreo de longitud=[*intervalo*].

Sintaxis:

```
vmstat [intervalo [número]]
```

Este comando no requiere de permisos especiales y no se cuenta a sí mismo como un proceso en ejecución. Los informes se utilizan para ayudar a eliminar cuellos de botellas. El comando *vmstat* toma información de los archivos /proc/meminfo, /proc/stat, /proc/*/stat.

Opciones:

intervalo	tiempo en segundos que trascurre entre refrescos. Si no se especifica muestra sólo un informe.
número	cantidad de informes a emitir, siempre y cuando esté definido el intervalo.
-a	muestra la cantidad de memoria activa e inactiva.

-s muestra una tabla con estadísticas de varios contadores de eventos.
-d informe de estadísticas de disco
-D resumen estadístico sobre la actividad del disco.
-p Dispositivo estadísticas detalladas sobre la partición.
-S *caracter* muestra la información según el *caracter* indicado, en las siguientes unidades: 1000 (k), 1024 (K), 1000000 (m), 1048576 (M).

Ejemplo

```
vmstat 60  5
```

El comando anterior generará 5 informes, uno cada minuto. En dicha salida podemos identificar los siguientes campos:

```
root@simred:/etc# vmstat 60 5
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0   2280 427672  62576 378264    0    0     3     3  206   86  4  4 92  0  0
 0  0   2280 443036  62680 392984    0    0   250     5  178  331 27 12 61  0  0
 1  0   2280 246648  63592 490020    0    0  1613   206  269  713 54 13 31  2  0
 0  0   2280 267872  63888 494076    0    0    64   135  216  559 36 15 49  0  0
 0  0   2280 259568  63908 494228    0    0     3     5   71  133  3  3 94  0  0
root@simred:/etc#
```

Procesos
r n° de procesos esperando su tiempo de ejecución.
b n° de procesos en espera ininterrumpible.

Memory
swpd cantidad de memoria virtual empleada (KB).
free cantidad de memoria inactiva (KB).
buff cantidad de memoria empleada como buffers (KB).
cache la cantidad de memoria utilizada como caché de páginas.

Swap
si cantidad de memoria intercambiada desde el disco (KB/s)
so cantidad de memoria intercambiada al disco (KB/s).

IO
bo bloques enviados a un dispositivo de bloques (bloques/s).
bi bloques recibidos desde un dispositivo de bloques (bloques/s).

System
in n° de interrupciones por segundo, incluyendo al reloj.
cs n° de cambios de contexto por segundo.

CPU
us el porcentaje de tiempo que el CPU ejecutó código de nivel del usuario.
sy el porcentaje de tiempo que el CPU ejecutó código de nivel del sistema.
id el porcentaje de tiempo que el CPU estaba desocupado.
wa tiempo gastado esperando IO.
st tiempo robado de una máquina virtual. Antes de Linux 2.6.11, desconocido.

Ejemplo 2

```
vmstat  -s -S M
```

```
root@simred:/etc# vmstat  -s -S M
         1007 M total memory
          755 M used memory
          437 M active memory
          278 M inactive memory
          251 M free memory
           62 M buffer memory
          483 M swap cache
          382 M total swap
            2 M used swap
          380 M free swap
       461983 non-nice user cpu ticks
           29 nice user cpu ticks
       406622 system cpu ticks
     11127766 idle cpu ticks
         6191 IO-wait cpu ticks
          215 IRQ cpu ticks
       118235 softirq cpu ticks
            0 stolen cpu ticks
       461524 pages paged in
       398816 pages paged out
            0 pages swapped in
          570 pages swapped out
     24951881 interrupts
     96413003 CPU context switches
   1554380602 boot time
        23874 forks
root@simred:/etc#
```

Muestra un resumen de la actividad de la memoria expresada en MBytes.

Ejemplo 3

```
vmstat -a
```

```
root@simred:/etc# vmstat -a
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free  inact active   si   so    bi    bo   in   cs us sy id wa st
 0  0   2280 257460 285260 448484    0    0     4     3  206   87  4  4 92  0  0
root@simred:/etc#
```

Muestra la cantidad de memoria libre, inactiva y activa.

dmesg	Muestra y manipula el almacenamiento intermedio de anillo del kernel. Es donde residen los mensajes de inicio.

Sintaxis:

```
dmesg [-c]
```

Opciones:

-c limpia el almacenamiento intermedio después de mostrar su contenido.

Ejemplo

```
dmesg | grep DMA
```

```
root@simred:/etc# dmesg | grep DMA
[    0.000000]   DMA      [mem 0x00001000-0x00ffffff]
[    0.000000]   DMA zone: 32 pages used for memmap
[    0.000000]   DMA zone: 0 pages reserved
[    0.000000]   DMA zone: 3998 pages, LIFO batch:0
[   19.648629] pci 0000:00:01.0: Activating ISA DMA hang workarounds
[   23.101571] ata1: SATA max UDMA/133 abar m8192@0xf0820000 port 0xf0820100 irq 11
[   23.106317] ata2: PATA max UDMA/33 cmd 0x1f0 ctl 0x3f6 bmdma 0xd000 irq 14
[   23.106318] ata3: PATA max UDMA/33 cmd 0x170 ctl 0x376 bmdma 0xd008 irq 15
[   23.281521] ata3.00: ATAPI: VBOX CD-ROM, 1.0, max UDMA/133
[   23.287936] ata3.00: configured for UDMA/33
[   23.427276] ata1.00: ATA-6: VBOX HARDDISK, 1.0, max UDMA/133
[   23.429359] ata1.00: configured for UDMA/133
root@simred:/etc#
```

En este ejemplo hemos filtrado las líneas que hacen referencia al uso de DMA y zona de memoria asignada al mismo.

Área de intercambio

El área de intercambio o *swap*, es la zona de memoria virtual en el disco duro, que permite albergar temporalmente páginas de memoria RAM. El área de intercambio puede ser una partición o un archivo y debe tener al menos 10 páginas. Linux permite 32 áreas de intercambio. El archivo */proc/swaps* nos muestra las áreas de swap activas en el sistema.

```
cat  /proc/swaps
```

```
root@simred:/etc# cat /proc/swaps
Filename                                Type            Size    Used    Priority
/dev/sda5                               partition       392188  2280    -1
root@simred:/etc#
```

Cada línea de salida nos muestra el nombre del archivo/dispositivo swap, el tipo de espacio swap, el tamaño total y la cantidad de espacio en uso (en kilobytes). La prioridad se utiliza cuando se usan múltiples archivos de espacio de intercambio. Cuanto más baja es la prioridad, más probable es que se use el archivo de intercambio. Si varios archivos de intercambio tienen la misma prioridad, las páginas de memoria se distribuirán como en un RAID[16] de nivel 0. Esto permite que los dispositivos de swap se usen en paralelo, cosa que puede aumentar la eficacia, sobre todo si están en discos independientes.

El archivo */etc/fstab* almacena información de cómo serán montadas e integradas al sistema las particiones de disco y sistemas de archivos. Por lo que, una vez creada el área de intercambio, se deberá agregar una línea a este archivo como sigue:

```
partition/file   swap  swap   defaults   0  0
```
[17]

Para que pueda utilizarse automáticamente desde el siguiente arranque del sistema.

[16] RAID 0 se usa normalmente para proporcionar un alto rendimiento de escritura ya que los datos se escriben en dos o más discos de forma paralela, aunque un mismo archivo solo está presente una vez en el conjunto.
[17] Este tema se trata en el capítulo de Entrada/Salida.

Los pasos a seguir para crear un área de intercambio tipo file, son:

1- Crear el dispositivo/archivo de intercambio (comando *dd*)
2- Preparar el dispositivo como swap (comando *mkswap*)
3- Escribir el archivo en el disco (comando *sync*)
4- Activar el área de intercambio para poder utilizarla (comando *swapon*)

dd convierte y copia un archivo.

Copia un archivo de la entrada estándar a la salida estándar, por omisión, con un tamaño de bloque seleccionable por el usuario, a la par que, opcionalmente realiza sobre él ciertas conversiones. Lee de la entrada un bloque cada vez, empleando el tamaño de bloque especificado para la entrada (el valor predeterminado es 512 bytes). El bloque de salida tiene la misma longitud que cuando se leyó a menos que se haya especificado la conversión *sync,* en cuyo caso los datos se rellenan con nulos (o espacios). Este comando generalmente es utilizado para realizar imágenes de discos/particiones.

Sintaxis:

```
dd  if=entrada  of=salida  bs=n  count=bloques
```

if=fichero es de donde se tomarán los datos, por ejemplo: /dev/zero (todos ceros), /dev/random (de forma aleatoria), /dev/sdb1 (una partición, para ser clonada).

of=salida nombre del archivo de salida.

bs=n cantidad de bytes que se lee y escribe n bytes de una vez, puede utilizarse K,M,G (Kbyte, Mbytes y Gbytes respectivamente).

count=bloques cantidad de bloques a copiar de tamaño determinados por *bs*, del archivo de entrada.

conv=CONVERSION convierte el archivo según se haya especificado en el o los argumentos.

Algunas conversiones permitidas son:

ascii	convierte EBCDIC a ASCII.
ebcdic	convierte ASCII a EBCDIC
lcase	cambia las letras mayúsculas a minúsculas.
ucase	cambia las letras minúsculas a mayúsculas.

mkswap prepara un dispositivo o archivo como área de intercambio. El del Sistema de archivos se puede especificar en bloques.

Sintaxis:

```
mkswap [opciones]  dispositivo  [tamaño]
```

El argumento *dispositivo* puede ser un archivo o una partición (ej. /dev/hda5). Al crear una partición de disco para ser utilizada como swap, debe identificarse como ID 82 tipo LINUX_SWAP. Si el *dispositivo* es un archivo, éste no debe tener agujeros, por lo que no podemos utilizar el comando cp. En este caso debemos utilizar el comando dd para crearlo.

El parámetro *[tamaño]* es el tamaño del área de intercambio, dada en bloques (1024 bytes por bloque). El tamaño total de memoria es un múltiplo entero del tamaño de la página de la máquina, el comando mkswap lee este valor del kernel.

Opciones:

-c comprueba si hay bloques defectuosos en el dispositivo antes de crear el sistema de archivos de intercambio.
-p, --pagesize SIZE específica el tamaño de página en bytes

sync	guarda el contenido del caché de disco dentro del disco físico. De esta forma se fuerza la escritura en disco de la información que ha cambiado.

swapon	des/habilita dispositivos o archivos para el paginado e intercambio.

El dispositivo de intercambio se activa con el comando *swapon dispositivo* y se desactiva con *swapoff dispositivo*. En el archivo /proc/swaps se muestra una lista de los dispositivos de intercambio activos, y en /etc/fstab se suelen añadir las particiones swap que se activan al inicio del sistema.

Sintaxis:

```
swapon [options] [specialfile...]
```

Opciones:

-p Especifica la prioridad del dispositivo de intercambio. Cuando la prioridad no está definida el valor predeterminado es -1.
-s, --summary Muestra el resumen de uso del swap por dispositivo.

Ejemplo

```
root@simred:/etc# swapon -s
Filename                                Type            Size    Used    Priority
/dev/sda5                               partition       392188  2280    -1
/etc/fichero-swap                       file            8188    0       -2
root@simred:/etc#
```

La opción –s muestra el contenido del archivo /proc/swaps.

Ejemplo

Crear un archivo para utilizarse como área de intercambio:

```
# dd  if=/dev/zero  of=fichero-swap  bs=1024  count=8192
# mkswap  fichero-swap  8192
# sync
# swapon  fichero-swap
```

Como habrá observado utilizamos el prompt del superusuario "#".

El comando *dd* lee desde el archivo */dev/zero* y lo envía al *fichero-swap*, lee y escribe *count* bloques de *bs* bytes por vez. Es decir, 8192 bloques de 1024 bytes cada uno. El archivo tendrá un tamaño de 8MB. El comando sync escribe todos los bloques copiados, al disco. Un área de

intercambio no debe tener *agujeros*, por esta razón utilizamos el comando dd y no el comando cp, para crear el archivo swap. Verificamos que el área de intercambio esté activa con *swapon –s.*

Con el comando *swapon* le indicamos al sistema que sobre el dispositivo fichero-swap, van a tener lugar las actividades de paginado e intercambio. Las llamadas a *swapon* se hacen normalmente en el archivo de inicio del sistema en modo multiusuario, normalmente */etc/rc*, haciendo ahí que todos los dispositivos de intercambio estén disponibles, de modo que las actividades de paginado e intercambio se intercalen entre varios dispositivos o archivos.

Archivo mapeado en memoria

Un archivo mapeado en memoria es una copia idéntica del archivo que está en el disco. Así pues, acceder a un archivo que está en memoria es mucho más rápido que acceder al caché de disco. Las operaciones de E/S sobre un archivo mapeado en memoria, evitan utilizar los buffers del kernel por lo que son más rápidas. A su vez, estos archivos se pueden utilizar también para compartir datos, es decir, varios procesos concurrentes pueden acceder al archivo mapeado en memoria. Los datos serán independientes de los procesos. El archivo /usr/include/asm/unistd tiene todas las llamadas al sistema, incluidas las de administración de memoria, como ser: mmap, munmap, sbrk, mlock, y munlok.

Las siguientes son algunas de las llamadas al sistema que administran el mapeo en memoria de los archivos:

mmap	Mapea un archivo de disco en memoria.
munmap	Elimina el mapeo de un archivo
msync	Salva el archivo mapeado, al disco.
mprotect	Limita el acceso a la imagen en memoria (protección).
mlockall	Evita que cualquier porción de la imagen en memoria sea intercambiada con su contraparte de disco.
munlockall	Elimina el bloqueo establecido por mlockall.

Necesitará utilizar las llamadas al sistema cuando decida implementar su propio método de administración de memoria.

Actividad 1

1- Utilice el comando free para dar un informe sobre la memoria usada y libre, en Mbytes y cada 75 segundos.

```
free  -m -s75
```

2- Sin suspender el proceso anterior y con el objetivo de ejecutar procesos lo suficientemente grandes como para reflejar cambios en los datos del archivo meminfo, desde el entorno gráfico ejecute alguna aplicación (ej. Editor). Regrese a modo texto y analice los nuevos informes de free. Observe que los valores en Free y en Shared han cambiado. ¿Por qué?

3- Genere con el comando utilizado en el punto 1, un archivo de informes y llámelo **monitor.**

```
free -m -s75  >  monitor
```

Luego de unos mitutos podemos ver su contenido y obtener un informe completo de cómo han ido cambiando los distintos espacios de memoria.

4- De un informe de los procesos que se están ejecutando en el sistema. ¿v*mstat* figura como proceso? ¿Por qué?

5- Muestre por pantalla la salida del comando top y compare con el comando free.

Actividad 2

1- Muestre por pantalla un informe sobre las particiones de swap, memoria swap utilizada y prioridades. Analice la información.

```
more   /proc/swaps
```

Este archivo mide el espacio swap y su uso. Para un sistema con tan sólo una partición de espacio swap, la salida de datos de */proc/swap* será:

Filename	Type	Size	Used	Priority
/dev/hda6	partition	136512	20024	-1

Mientras que alguna de esta información se puede encontrar en otros archivos en el directorio */proc/, /proc/swap* proporciona una instantánea rápida de cada nombre de archivo swap, tipo de espacio swap, el tamaño total, y la cantidad de espacio en uso (en kilobytes). La columna de prioridad es útil cuando múltiples archivos swap están en uso. Cuanto más baja es la prioridad, más probable es que se use el archivo swap.

2- Genere un archivo de control que contenga un informe completo sobre los procesos (listos, inactivos, etc), memoria utilizada y paginada. Realice la tarea cada 30 seg. y por el lapso de 5 minutos. Realice esta tarea en background.

```
vmstat   30  10  >  control&
```

3- Luego de este lapso analice la información del archivo **control.**

```
more   control
```

4- Genere un archivo en su directorio con el nombre **intercambio**, de tamaño 2Mbytes para swap que intercambie bloques de 2kB.

```
# dd  if=/dev/zero  of=intercambio  bs=2048  count=1024
# mkswap intercambio  1024
# sync
# swapon intercambio
```

5- Repita el punto 1.

```
more  /proc/swap
```

Verá que hay una nueva unidad tipo file para intercambio.

6- Indique una nueva prioridad para el archivo de intercambio para que sea la última área en ser utilizada, por ejemplo si las prioridades de las áreas en uso, están entre -1 y -3 podría utilizar el valor 1.

```
swapoff  intercambio
```

Como en el punto 4 el área fue activada, debe primeo desactivarla con el comando swapoff y recién después podrá cambiar la prioridad.

```
swapon –p 1 intercambio
```

Actividad 3

1. Linux utiliza para la administración de la memoria:
a) Intercambio
b) Prepaginación
c) Paginación por demanda
d) Particiones fijas
e) Ninguna de las anteriores es correcta.

2. El comando **free** nos muestra datos que los obtiene del archivo:
a) /proc/swap
b) /proc/meminfo
c) /dev/hda3
d) /proc/1276/mem
e) Ninguna de las anteriores es correcta.

3. El área de swap de un sistema LINUX se utiliza como soporte en disco para:
a) Respaldo del file system
b) Memoria virtual
c) Tabla de i-nodos
d) Boot del sistema
e) Ninguna de las anteriores es correcta.

4. ¿Qué comando o utilidad usaría si deseara conocer información estadística del uso de la memoria, paginación, etc.?

5. Genere un archivo llamado **control** en el que se genere un informe sobre la memoria usada y libre cada 90 segundos. Luego analice la información obtenida.

6. La memoria virtual en Linux, es:
   a) Respaldo del sistema de archivos.
   b) El área de disco donde se almacena el superblock.
   c) Necesariamente una partición de disco.
   d) El área de intercambio (swap) en el disco.
   e) E. Ninguna de las anteriores es correcta.

Capítulo 10

Entrada / Salida

La Administración de los dispositivos es una función muy importante del SO. Consiste en el control, asignación y uso de los recursos de hardware conectados a la computadora. Sean estos: disco, pendrive, CD, impresora, escáner, mouse, teclado, etc. Linux maneja a estos dispositivos como archivos especiales que se integran al File System al igual que cualquier otro archivo y poseen su ruta de acceso en el subdirectorio */dev*[18]. El archivo de dispositivo representa al dispositivo para comunicarlo con programas que se ejecutan en la máquina. Para ello debe existir un driver apropiado para cada dispositivo.

Las aplicaciones pueden abrir, leer y escribir los archivos especiales de la misma forma que lo hacen con un archivo común o regular. Esto se convierte en una ventaja para la seguridad y protección de los dispositivos de E/S, restringiendo el acceso a estos archivos a los distintos usuarios, si es necesario.

Esta flexibilidad capaz de abstraer el dispositivo y considerar solo lo fundamental, la comunicación, le ha permitido adaptarse a la rapidez de los cambios y a la variedad de dispositivos que existen en la actualidad.

Debemos tener presente que para Linux todo dispositivo físico es representado o asociado a un archivo, ya sean discos duros, CD-ROM, memorias usb, dispositivos de comunicación como puertos seriales y paralelos, módems, incluso también las terminales son dispositivos asociados o enlazados (linked) a un archivo. Estos archivos se encuentran bajo el directorio /dev y se los denominan archivos especiales.

Los archivos especiales[19] se clasifican en:

1. b – block devices: dispositivos orientados a bloques
2. c – character devices: dispositivos orientados a caracteres
3. s – socket devices: dispositivos orientados a sockets

Los *archivos especiales **de bloque*** disponen de una memoria intermedia o buffer para evitar que el programa o recurso que los requiere, ya sea hardware o software, se quede sin datos durante una transferencia (entrada/salida). Se utilizan para los discos u otros dispositivos de almacenamiento de datos, como CD-ROM, DVD, etc. Estos dispositivos constan de una serie de bloques numerados, y cada bloque se puede direccionar y acceder individualmente en las operaciones de lectura/escritura.

Los *archivos especiales **de carácter*** utilizan la transmisión serial de datos, sin usar buffer y se usan para las terminales, impresoras, ratones, plotters y demás dispositivos de E/S que aceptan o producen un flujo de caracteres. Estos archivos no permiten un acceso aleatorio a los mismos.

Un ***socket*** de Internet no es un archivo de un dispositivo, sino una forma de comunicación entre procesos**.** Por ejemplo, el archivo /dev/log representa al Socket de syslog.

Los dispositivos en Linux son identificados con un designador de dos o tres letras, además si el dispositivo admite particiones se utiliza una progresión numérica o alfabética para identificar la partición. En la siguiente tabla se indica el designador de tipo dispositivo y su descripción.

Ejemplos de archivos de dispositivos orientados a bloques

Nombre del archivo	**Significado**
fd0	disquetes
hda	IDE-1er Disco duro. En FreeBSD el archivo es ad0.
hdb	IDE-2do. Disco duro
hda1	1ra. partición de disco del primer IDE-Disco duro. En FreeBSD es ad0s1
hda15	Partición lógica nro. 15 del 1er disco duro IDE
sda	1er. Disco duro SCSI con el menor SCSI-ID
sdb	2do. Disco duro SCSI con el siguiente SCSI-ID

[18] En el caso de Solaris en /devices se encuentra los dispositivos de la máquina, ya sean reales, como un disco duro, o virtuales, como ***/dev/null***.

[19] El archivo */proc/devices* muestra los diversos dispositivos de caracteres y de bloque actualmente configurados.

sda1	1ra. Partición del primer disco duro SCSI
sda15	partición lógica del primer disco duro SCSI
scd0	1er lector SCSI-CD-ROM

Ejemplos de archivos de dispositivos orientados a caracteres[20]

Nombre del Archivo	Significado
ttyS0	1er. puerto serie, antiguamente el teclado del Terminal
lp0	1er. puerto paralelo
lp1	2do. puerto paralelo
ttyX	Enlace simbólico para un Pseudoterminal
usbdev1.1	Archivo de dispositivo para aparatos con USB
mouse	Enlace simbólico al ratón

Linux y la gestión de discos

Dentro de una organización, una de las responsabilidades asociadas al departamento de TI, es la de gestionar y administrar los dispositivos de almacenamiento, así como verificar su estado y rendimiento. Entre las diferentes tareas que se deben realizar se encuentran las asociadas al particionado, formateo y montaje de dispositivos o discos.

Por distintos motivos, se hace necesario reorganizar o crear particiones en un disco, agregar más espacio a la unidad del sistema, reformatear discos, o montar y desmontar dispositivos para lograr el funcionamiento deseado en el sistema, independientemente de la 'distro' que se esté utilizando.

Para su seguridad en recomendable disponer con, al menos, dos particiones, una para archivos del sistema y otra para los datos de los usuarios, de esa forma ante un fallo del sistema, se asegura que no se afectarán sus archivos personales. De la misma manera, si tenemos que volver a instalar desde cero el sistema operativo, solo será necesario formatear su partición, y las particiones adicionales que tengamos creadas para nuestras copias de seguridad y archivos personales estarán resguardados. Además cada dispositivo de disco, aloja un sistema de archivos que, conceptualmente, no es más que un árbol de directorios que puede ser integrado en el árbol único de Linux utilizando el montaje de dispositivos.

Particiones de Disco

Se puede decir que una partición es una sección lógica del disco y a nuestra apariencia funciona como si fuera un disco duro independiente. Al crear una partición, se segregan[21] un conjunto de sectores (unidad mínima de datos en un disco) y se les asigna una identificación o nombre de volumen.

En GNU/Linux puede haber hasta cuatro particiones **Primarias**, o hasta tres primarias y una extendida. Una partición **Extendida** es aquella cuyo contenido es a su vez particionado en varias particiones lógicas. Como mínimo debemos definir en una partición extendida, una partición lógica.

Las cuatro particiones primarias/extendida en Linux, reciben los números del 1 al 4, estén presentes o no, en cuanto a las particiones lógicas empiezan con el número 5 en adelante.

Como ya sabemos en Linux, los discos como todos los dispositivos se representan y referencian por archivos, estos se encuentran en el directorio **/dev**.

[20] La tarjeta de red, por ejemplo Ethernet, ISDN, no es contactada a través de archivos de dispositivo, sino a través de TCP/IP.
[21] Segregar: Separar una cosa de otra de la que forma parte para que siga existiendo con independencia

Los nombres de discos rígidos serán:

/dev/hd[a-h] para discos IDE
/dev/sd[a-p] para discos SCSI
/dev/ed[a-d] para discos ESDI
/dev/xd[ab] para discos XT

El nombre del dispositivo se refiere al disco entero. Por lo que la partición es un nombre de dispositivo seguido por un número de partición. Por ejemplo, /dev/hda1 es la primera partición del primer disco duro IDE en el sistema. Los discos IDE pueden tener hasta 63 particiones, los SCSI hasta 15.

A la hora de instalar un Linux se necesita al menos dos particiones de disco, una donde irá el sistema operativo y programas, y otra llamada swap o partición de intercambio[22], aunque también puede usar archivos swap, las particiones son más eficientes.

Por razones de administración, copias de seguridad o pruebas, se puede utilizar más particiones de las mínimas recomendadas anteriormente.

El Proceso de Particionar

Si no se poseen varios discos rígidos, y se necesita particionar el único disco disponible, se recomienda utilizar *fdisk* una utilidad de la línea de comandos de Linux, que está basada en texto y es guiado por menú para administrar particiones de disco duro a través de la manipulación de las tablas de partición.

La información sobre cómo se particiona un disco, se almacena en su primer sector. Este primer sector es el llamado *registro de arranque maestro (MBR)* del disco. Técnicamente, los primeros bytes del MBR contienen un código que se ejecuta en modo real y que explora la tabla de particiones buscando la Partición Activa, que debe contener un sistema de archivos y los archivos necesarios para iniciar la ejecución y carga del sistema operativo. Sólo una partición puede estar apuntada como Activa, a pesar de que puede haber más de una partición conteniendo un sistema operativo[23].

Utilizando ***fdisk*** se puede crear un máximo de 4 particiones primarias o hasta 3 primarias y un número mayor de particiones lógicas en 1(una) partición extendida.

En resumen, un disco rígido posee una MBR (donde hay reservado espacio para cuatro estructuras de datos), la tabla de partición y las particiones propiamente dichas.

En el proceso de *particionado* se escriben los sectores que conformarán la tabla de partición (la cual contiene información acerca de la partición: tamaño en sectores, posición con respecto a la partición primaria, tipos de partición existentes, sistemas operativos instalados, etc)

fdisk	permite crear, modificar o eliminar particiones en el disco rígido manipulando la tabla de particiones.

Se debe tener presente que la utilidad *fdisk* requiere ejecutar como administrador (root) para que funcione. En sistemas basados en Debian y Ubuntu, se puede usar ***sudo***. En otras distribuciones, usar el comando ***su*** para obtener una shell de root y luego los comandos.

Sintaxis:

```
fdisk [opciones] [dispositivo]
```

[22]Partición Swap o de Intercambio permite compensar las limitaciones de la memoria física, tema tratado en el capítulo de Administración de Memoria.

[23]Los Gestores de Arranque son unos programas que dan la opción al usuario de seleccionar uno de los múltiples S.O. instalados en el disco para que realice el arranque. Ej: GRUB, LILO.

En *dispositivo* se indica el nombre del dispositivo o disco a particionar, en cambio si no pasamos un argumento, éste seleccionará la primera unidad de disco que encuentre.

En opciones se deberá teclear la letra correspondiente a la opción elegida, para simplificar esta tarea *fdisk* proporciona la posibilidad de ejecutar el comando sin opciones, indicando solo el dispositivo y la utilidad le devuelve por la salida estándar el siguiente menú de ayuda, de manera que el usuario elija la opción.

Sin opciones ni argumentos, *fdisk* muestra todos los discos/particiones definidas, indicando cantidad de cilindros, sectores, nodos-i, etc.

Al iniciar el proceso de particionado podemos visualizar todas las opciones de los comandos disponibles de fdisk, como sigue:

```
# sudo fdisk /dev/hda

Comando (m para ayuda): m
   Orden      Acción de comando:
    a   alternar un indicador de arranque
    b   editar bsd
    c   alternar el indicador de compatibilidad DOS
    d   eliminar una partición
    l   lista los tipos de partición conocidos
    m   imprime el menú
    n   añadir una nueva partición
    o   crear una nueva tabla de particiones DOS vacía
    p   imprimir la tabla de particiones
    q   salir sin guardar los cambios
    s   crear una nueva etiqueta de disco Sun vacía
    t   cambiar el id del sistema de una partición
    u   cambiar las unidades de visualización / entrada
    v   verificar la tabla de particiones
    w   escribe la tabla en el disco y salir
    x   Funcionalidad adicional (sólo expertos)

Orden (m para obtener ayuda):__
```

Ejemplo 1

Si deseamos particionar el primer disco IDE o sea /dev/hda, lo primero que debemos hacer es visualizar el dispositivo que nos interesa particionar, eligiendo la opción ‘l’ del menú anterior o saliendo del menú y tipeando:

```
# sudo fdisk -l /dev/hda
```

Se obtiene la siguiente salida:

```
Disk /dev/hda: 15 heads, 57 sectors, 790 cylinders
Units = cylinders of 855 * 512 bytes
Device Boot    Begin   Start     End  Blocks   Id  System
/dev/hda1          1       1      24   10231+  82  Linux swap
/dev/hda2         25      25      48   10260   83  Linux native
/dev/hda3         49      49     408  153900   83  Linux native
/dev/hda4        409     409     790  163305    5  Extended
/dev/hda5        409     409     744  143611+  83  Linux native
/dev/hda6        745     745     790   19636+  83  Linux native
```

En el primer sector de la partición extendida (/dev/hda4) se hallará un registro maestro que indicará como mínimo, desde qué sector hasta qué sector del HD se comprenden cada una de las particiones lógicas (/dev/hda5 y /dev/hda6). La única diferencia es que los datos de localización lógica se hacen relativos al primer sector de la partición y no al primer sector del disco.

Ejemplo 2

En el caso que se necesite combinar algunas particiones para formar una más grande, primero se tendrá que eliminar esas particiones. En este caso del ejemplo se eliminarán las particiones /dev/hda5 y /dev/hda6, que son particiones lógicas y la /dev/hda4 partición extendida.

```
# sudo fdisk /dev/hda
```

- ✓ Aparece el menú de opciones y se tipea 'd' para eliminar una partición.

```
Comando (m para ayuda):
 Orden      Acción de comando:
  a   alternar un indicador de arranque
  ...
  d   eliminar una partición
  ...
   w   escribe la tabla en el disco y salir
 Orden (m para obtener ayuda): d
```

- ✓ Pedirá que especifique 'p' para una partición primaria o 'e' para una partición extendida.
- ✓ El próximo mensaje pedirá que ingrese el número de partición.

```
Orden (m para obtener ayuda): d
Command action
e  extended
p  primary partition (1-4)
e
Partition number  (5-6) :6
```

- ✓ Introducimos 5 en nuestro caso, para eliminar /dev/hda6 y luego hacemos lo mismo para /dev/hda5 y la /dev/hda4.
- ✓ Guardamos los cambios introduciendo 'w' en el comando, y reinicia los cambios.
- ✓ Luego con la opción del menú ´p´ vemos como quedó nuestra tabla de particiones.

```
Disk /dev/hda: 15 heads, 57 sectors, 790 cylinders
Units = cylinders of 855 * 512 bytes
   Device Boot    Begin   Start     End  Blocks   Id  System
   /dev/hda1        1       1      24   10231+  82  Linux swap
   /dev/hda2       25      25      48   10260   83  Linux native
   /dev/hda3       49      49     408  153900   83  Linux native
```

Ejemplo 3

Luego de eliminar las particiones deseadas procedemos a crear una nueva partición de disco con un tamaño específico. Para ello, se ingresa el comando ´n´. El mensaje pedirá que especifique si se desea crear una partición primaria o una partición extendida. Escribe 'p' para primaria o 'e' para extendida y a continuación el número que se asignará a la partición.

```
Orden (m para obtener ayuda): n
Command action
e  extended
 p  primary partition (1-4): p
Partition number  (1-4) :4
```

- ✓ A continuación, se pedirá que introduzca el primer número de cilindro o sector de la partición que se va a crear. Se puede presionar Enter para aceptar los valores predeterminados.

- ✓ Se pedirá que ingrese el último cilindro o número de sector de la partición que se va a crear, o presionar Enter para usar todo el espacio disponible después del primer sector o ingresar un tamaño específico como +2G o +256M por ejemplo para una partición de 2 gigabytes o 256 megabytes, respectivamente.
- ✓ Es conveniente revisar los cambios realizados con el comando ´p´.

```
Device Boot    Begin   Start     End  Blocks   Id  System
/dev/hda1          1       1      24   10231+  82  Linux swap
/dev/hda2         25      25      48   10260   83  Linux native
/dev/hda3         49      49     408  153900   83  Linux native
/dev/hda4        409     409     790  163305   83  Linux
```

- ✓ A continuación se debe asignar el tipo de ID que tendrá esa partición. Es posible que el tipo deseado sea el asignado por defecto o que necesitemos cambiarlo, para ello veamos a continuación el siguiente tema.

Tipos de particiones

Las tablas de particiones tienen un byte por cada tipo partición (Columna ID del ejemplo anterior), que identifica el sistema operativo que utiliza esa partición.
La siguiente lista (hexacode) está disponible en la utilidad de Linux **fdisk** con la opción ´t´

```
 0  Empty            1e  Hidden W95 FAT1  80  Old Minix
 1  FAT12            24  NEC DOS          81  Minix / old Lin
 2  XENIX root       39  Plan 9           82  Linux swap / So
 3  XENIX usr        3c  PartitionMagic   83  Linux
 4  FAT16 -32M       40  Venix 80286      84  OS/2 hidden C:
 5  Extended         41  PPC PReP Boot    85  Linux extended
 6  FAT16            42  SFS              86  NTFS volume set
 7  HPFS/NTFS        4d  QNX4.x           87  NTFS volume set
 8  AIX              4e  QNX4.x 2nd part  88  Linux plaintext
 9  AIX bootable     4f  QNX4.x 3rd part  8e  Linux LVM
 a  OS/2 Boot Manag  50  OnTrack DM       93  Amoeba
 b  W95 FAT32        51  OnTrack DM6 Aux  94  Amoeba BBT
 c  W95 FAT32 (LBA)  52  CP/M            9f  BSD/OS
 e  W95 FAT16 (LBA)  53  OnTrack DM6 Aux  a0  IBM Thinkpad hi
 f  W95 Extd (LBA)   54  OnTrackDM6       a5  FreeBSD
 10  OPUS           55  EZ-Drive         a6  OpenBSD
 11  Hidden FAT12   56  Golden Bow       a7  NeXTSTEP
 12  Compaq diagnost 5c  Priam Edisk     a8  Darwin UFS
 14  Hidden FAT16 -3 61  SpeedStor       a9  NetBSD
 16  Hidden FAT16   63  GNU HURD or Sys  ab  Darwin boot
 17  Hidden HPFS/NTF 64  Novell Netware   b7  BSDI fs
 18  AST SmartSleep  65  Novell Netware   b8  BSDI swap
 1b  Hidden W95 FAT3 70  DiskSecure Mult  bb  Boot Wizard hid
 1c  Hidden W95 FAT3 75  PC/IX
```

Si se desea cambiar el tipo de partición la opción del menú es la ´t´, en caso contrario procedemos a ejecutar directamente el próximo paso que es el comando ‘w’ para escribir los cambios y reiniciar el sistema.

```
Command (m for help): t
Partition number (1-4): 4
Hex code (type L to list codes): 86
Changed system type of partition 2 to 82 (NTFS volumen set)

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

Con esto, ya se puede utilizar el comando fdisk para manejar las particiones Linux.

Formatear una partición

Dentro de una partición puede que no haya ninguna estructura y entonces no es utilizable por ningún sistema operativo, o puede haber un sistema de archivos, que es la estructura para organizar ese espacio físico de almacenamiento para que un sistema operativo pueda utilizarlo.

Definir un tipo de partición no le da el formato o estructura necesaria para almacenar físicamente los datos que deseamos, por lo que después de crear la partición, se debe formatearla con un sistema de archivos.

Con el paso del tiempo Linux fue evolucionando en distintos sistemas de archivos:

ext – Extended file system (sistema de archivos extendido)
ext2 – Second extended file system (segundo sistema de archivo extendido)
ext3 – Third extended file system (tercer sistema de archivos extendido), una forma de ext2 con registro en diario
ext4 – Fourth extended file system (cuarto sistema de archivos extendido), una mejora de ext3 y también un sistema de archivos con registro en diario con soporte para extensiones.

> ***mkfs*** se utiliza para dar formato a un dispositivo de almacenamiento de bloque con un determinado sistema de archivos

Sintaxis:

```
mkfs  [opciones]  dispositivo
```

Opciones:

-v Produce una salida detallada, incluyendo todas las órdenes específicas del sistema de archivos que se ejecutan.
-t fstype Especifica el tipo de sistema de archivos que se construirá. Si no se especifica, se utiliza el tipo de sistema de archivos por defecto (actualmente ext4).
-c Comprueba el dispositivo de bloques malos antes de construir el sistema de archivos.
-h, – help Pantalla de ayuda y sale.

Ejemplo 1

```
# mkfs.ext4  /dev/sda2
```

Este comando crea un *sistema de archivos ext4 para Linux* en un disco SCSI en su partición número 2. Luego para poder acceder a ella, se debe montar el sistema de archivos que contiene con el comando *mount.*

Ejemplo 2

```
# mkfs  -t  ext4  /dev/sdd1
# mkfs.ext4  /dev/sdd1
# mkfs  /dev/sdd1
```

En los tres casos crea un sistema de archivo en formato ext4 en una partición SCSI.

Ejemplo 3

```
# mkfs.vfat  /dev/hda1
```

Crea un sistema de archivos vfat en la partición hda1

Ejemplo 4

```
# mkfs.msdos  /dev/hda1
```

Formatea una partición hda1 en formato MS-DOS

Ejemplo 5

```
# mkfs -t vfat 32 -F /dev/hda1
```

Crear un archivo de sistema FAT32 en hda1

Ejemplo 6

```
# mkfs.vfat -n nombre /dev/sdb1
```

Ejemplo de mkfs para formatear una memoria USB. Donde *vfat* es el formato que le vamos a dar a la memoria (FAT32, es el más común), nombre: es el nombre que va a tener la memoria USB. Ejemplo MemUsby /dev/sdb1 es la ubicación.

Montar y Punto de montaje

Como ya vimos en Linux todo es un archivo y tanto el hardware como el software está almacenado como un archivo de texto, por lo tanto, no depende de si el equipo tiene 1, 3 o 5 discos duros para crear las unidades c:\, d:\ ó f:\.

En Linux estamos totalmente abstraídos del concepto de unidades, su estructura lógica, es independiente de la estructura del hardware, lo cual permite que un sistema de archivos forme parte integral de la estructura de otro, en una dirección específica del filesystem que es denominado *Punto de montaje.*

Para poder acceder a un sistema de archivos es necesario identificarlo como de un tipo concreto, y es de allí que nace el concepto de "montar" o "desmontar" un dispositivo. En síntesis, el comando *mount* de Linux, abstrae el hardware haciendo que una unidad de disco u otro dispositivo se asocie al sistema de archivos en un determinado directorio.

Todo el sistema Linux tiene por origen la raíz o root, representada por /, y bajo ese directorio se localizan todos los demás archivos accesibles en el sistema operativo.

Estos directorios que serán *Punto de Montaje* de un dispositivo y deberán ser creados previamente al montaje del dispositivo.

mount	monta un dispositivo o muestra un listado de ellos.

Sintaxis:

```
mount opciones dispositivo dir
```

Este comando le dice al kernel que al sistema de archivos que encuentre en `dispositivo` (que se refiere a algunas de las particiones, discos duros o dispositivos que vamos a montar) lo anexe al directorio `dir`, y mientras este sistema de archivos permanezca montado, el camino de `dir`, también llamado *Punto de Montaje*, hace referencia a la raíz del sistema de archivos en `dispositivo`. En el campo opciones se introducen datos como el tipo de sistema de archivos, si va a ser montado de sólo lectura, etc.

Opciones:

-a Monta todos los sistemas de archivos (del tipo dado) mencionados en el archivo /etc/fstab.
-f Causa que todo se realice excepto la llamada al sistema real; en otras palabras, realice un montaje ficticio. Se utiliza para comprobar si el montaje se realizaría correctamente.
-n Monta el dispositivo sin escribir en el archivo /etc/mtab. Esto es posible cuando el sistema de archivos es de uso temporal o cuando /etc está en un sistema de archivos de lectura exclusiva.
-t *Tipo de dispositivo fat, ext3, ext4, etc Directorio de destino.* El argumento que sigue a -t se emplea para indicar el tipo del sistema de archivos.
-r Monta el sistema de archivos como sólo lectura.
-w Monta el sistema de archivos para lectura/escritura (opción por defecto).
-o Las opciones se especifican mediante la bandera –o seguida por una ristra de opciones separadas por comas. Las siguientes opciones se aplican a cualquier sistema de archivos que se esté montando.
defaults Emplea las opciones predeterminadas: **rw, suid, dev, exec**, **auto, nouser, async.**
dev Interpreta dispositivos especiales de caracteres o bloques en el sistema de archivos.
noauto Sólo puede montarse explícitamente (esto es, la opción -a no hará que el sistema de archivos se monte).
ro Monta el sistema de archivos de lectura exclusiva.
rw Monta el sistema de archivos de lectura y escritura.
suid Permite el efecto de los bits SUID y SGID.
sync Toda la E/S al sistema de archivos debería hacerse sincrónicamente.
user Permite a un usuario ordinario montar el sistema de archivos. Esta opción implica las opciones noexec, nosuid y nodev (a menos que se indique explícitamente exec, suid y dev).

Ejemplo 1

```
#mount (sin parámetros)

sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs
(rw,relatime,size=10240k,nr_inodes=211058,mode=755)
devpts on /dev/pts type devpts
(rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /run/shm type tmpfs
(rw,nosuid,nodev,noexec,relatime,size=1631420k)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc
(rw,nosuid,nodev,noexec,relatime)
```

Muestra la lista de dispositivos montados, los cuales se encuentran en /etc/mtab. Una forma de que muestre lo mismo pero más ordenado es ejecutando: # mount | column -t

En principio, es necesario ser **root** para utilizar dicho comando (salvo que se use la opción “user”). Esto es así para garantizar la seguridad del sistema. No es aconsejable que usuarios normales estén montando y desmontando sistemas de archivos.

Ejemplo 2

```
# mount -n  /dev/sda5  /datos
# mount -rw /dev/sda5 /datos
```

El primer comando indica que el dispositivo sda5 se monte en el punto de montaje /datos de forma temporal sin afectar el archivo **/**etc/mtab.

El segundo caso indica que el dispositivo sda5 se monte en el punto de montaje /datos con permiso de lectura y escritura pero grabando los cambios en el archivo de configuración /etc/mtab**.**

Montar sistemas de archivos en forma automatizada

El comando *mount –av,* usualmente ejecutada en un guión de arranque hace que todos los sistemas de archivos mencionados en el archivo /etc/*fstab* se monten como se indique (excepto aquellos cuya línea contenga la palabra clave *noauto)*.

El archivo /etc/fstab contiene información sobre los sistemas de archivos instalados. Durante el arranque, el sistema a través del demonio *init* procesa este archivo y monta automáticamente los volúmenes correspondientes. Además de mount los comandos *umount* y *fsck* utilizan la información contenida en él.

El administrador del sistema es responsable de la creación y mantenimiento de este archivo. Cada sistema de archivos se describe en una línea y el orden de las líneas es importante, ya que el archivo es siempre recorrido de forma secuencial, y para montar algunas particiones es necesario haber montado previamente otras.

A continuación se puede ver un ejemplo de un archivo /etc/fstab y luego la explicación de cada elemento.

```
# file system    mount point      type       options     dump    pass
/dev/sda1           /boot         ext4      defaults       1      2
/dev/hda2            /            ext2      defaults       1      1
/dev/hda3            /usr         ext2      defaults       0      0
/dev/hda4           swap          swap      swap           0      0
/dev/sda6           /home         ext4      defaults       0      0
/dev/cdrom       /mnt/cdrom     iso9660     auto           0      0
/proc              /proc          proc      none           0      0
```

Cada una de las líneas del archivo representa un filesystem distinto, con los siguientes campos:

File system	Indica el dispositivo o la partición donde se encuentra el filesystem.
Mount point	Punto de montaje para el dispositivo especificado. El directorio desde el cual se accede al sistema de archivos del dispositivo montado.
Type	Tipo de sistema de archivos montado. Puede tomar varios valores, entre los que se destacan: ext2, ext3, ext4, iso9660, nfs, ntfs, reiserfs, smbfs, swap, vfat, xfs.
Options	Opciones para el montaje del filesystem (mencionadas anteriormente).
Dump	Indica a la utilidad dump si debe o no hacer backup del filesystem. Puede tomar dos valores: 0 y 1. Con 0 se indica que no se debe respaldar, con 1 que sí se haga. Lógicamente, depende de que se tenga instalado y configurado dump[24], por lo que en la mayoría de los casos este campo es 0.
Pass	Una indicación para el fsck (comando que chequea el filesystem) y nuevamente se define con un valor numérico. Las posibilidades son 0, 1 y 2. El 0 indica que el filesystem no debe ser chequeado, mientras que el 1 y el 2 le dicen a fsck que sí lo chequee. La diferencia es que el 1 representa una prioridad mayor que el 2, por lo que debe utilizarse para el sistema raíz y el 2 para el resto de los sistemas de archivos.

[24] vea man dump. Permite realizar copias de seguridad a sistemas de archivos o archivos y directorios.

El archivo /etc/*fstab* contiene una entrada especial para el sistema de archivos /proc. Este sistema de archivos /proc se utiliza para almacenar información acerca de los procesos del sistema, memoria disponible, y otros datos del mismo tipo. Si /proc no está montado, no funcionan comandos como *ps* u otro administrador de procesos.

Cuando se monte un sistema de archivos mencionado en /etc/fstab, basta con dar sólo el dispositivo o el punto de montaje.

Ejemplo

```
$ mount  /dev/hda3
```

Normalmente, sólo el súper usuario (root) puede montar sistemas de archivos. Sin embargo, cuando *fstab* contiene la opción *user*, cualquiera puede montar el sistema de archivos correspondiente.

También es necesario decir a Linux que se va a dejar de utilizar un sistema de archivos, para esto se utiliza la orden *umount.*

umount	desmonta un dispositivo o elimina los dispositivos instalados.

Sintaxis:

```
umount [opciones] dispositivo dir
```

El comando *umount* despega de la jerarquía o árbol de archivos el o los sistemas de archivos mencionados. Un sistema de archivos se puede especificar dando el directorio donde ha sido montado, o dando el dispositivo o archivo especial donde reside.

Un sistema de archivo no pude ser desmontando si está ocupado, un ejemplo claro es cuando hay en él archivos abiertos o cuando un proceso tiene su directorio de trabajo allí.

En casos de emergencia extrema, puede utilizar la opción –f para forzar que sea desmontando. Sin embargo si lo hace el programa que está utilizando esos archivos no terminará debidamente y puede perder parte de la información, por lo que se advierte que no lo haga nunca ya que es muy peligroso.

Los comandos *mount* y *umount*, mantienen una lista de los sistemas de archivos montados actualmente en el archivo /etc/mtab.

Los archivos /etc/mtab y /proc/mounts tiene contenidos muy similares.

Dispositivos Extraíbles - Memorias USB

Las diferentes distribuciones detectan automáticamente los Dispositivos Extraíbles como las memorias USB, ya que montan el dispositivo y crean un ícono por ejemplo “USB” en el escritorio. La ruta o camino por defecto del montaje de estas memorias es en el subdirectorio /mnt o /media.

En todos los casos, es posible instalar la memoria USB manualmente, como se hacía antes, gracias a la línea de comandos. Los pasos a seguir para que Linux reconozca una memoria USB son simples.

Ejemplo

Crear el directorio y montar la memoria

```
# mkdir /media/usb
# mount –t vfat /dev/sdb /media/usb
```

Nota: Si sdb no funciona, intentar con sdb1.

En el directorio /dev es posible verificar cuáles dispositivos existen en el sistema. Con el parámetro vfat estamos especificando que el sistema de archivos es del tipo FAT, también podría ser ext4, NTFS, etc.

Para montar la memoria USB fácilmente, sólo se tiene que agregar una línea en el archivo /etc/fstab que se ejecuta automáticamente al iniciar el sistema.

Siempre es necesario desmontar la memoria USB antes de retirarla del sistema, sino se corre el riesgo de dañarla en caso de una transferencia de datos.

Soporte de Dispositivos en Linux

El usuario no se comunica con los dispositivos directamente, es el kernel el encargado de comunicarse con el dispositivo y hacerlo funcionar, pero para ello el dispositivo debe estar soportado por el kernel.

El sistema de archivos /*proc contiene* los dispositivos reconocidos por el kernel de Linux y en el archivo */proc/devices* se encuentran documentados*,* es decir este archivo contiene una lista de los dispositivos de carácter y de bloque soportados y para los que existen controladores disponibles en el kernel con sus números correspondientes.

Los archivos /proc/interrups, /proc/ioports, /proc/pci, proc/ide, proc/scsi contienen información que varía desde las interrupciones (IRQ) utilizadas por los dispositivos, los puertos de I/O y los dispositivo PCI, IDE y SCSI del sistema.

El soporte de dispositivos de CD-ROM en el kernel de Linux va más allá de solo el hardware, incluye el sistema de archivos estándar para los CD, llamado ISO9660

El soporte del CD-ROM también incluye las unidades de CD-R (grabación) y CD-RW (de reescritura).

Información sobre el espacio del disco

El administrador del sistema (root) debe controlar constantemente todos los sistemas de archivos (file system) para asegurarse de que no se están llenando, ya que esto podría provocar errores de aplicación, errores del sistema, e incluso corrupción de archivos.

Para saber qué porcentaje de una partición se está utilizando, se puede usar el comando **df** (disk free space)**.**

df	se utiliza para ver los sistemas de archivos montados y el número de bloques de espacio del disco libres que tiene cada uno.

Sintaxis:

```
df [ - opciones]
```

Si se ingresa el comando df sin opciones se obtendrá una salida como ésta:

```
File system      1k-Bloques      Usados        Libres      %usado  Pto.montaje
/dev/root        1433600        1112914       320686       78%         /
/dev/boot         30720           17606        13114       58%       /stand
/dev/u           2494444         2197484       296960       89%         /u
```

Opciones:

- **.** (punto) Muestra el espacio que está utilizando el sistema actual solamente.
- **-i** Obtendremos información acerca de los i-nodos ocupados y libres en cada partición como se muestra a continuación:
- **–a** o **all** nos muestra sistemas de archivos, aunque éstos tengan cero bytes.

Ejemplo 1

```
$df -i
   S.ficheros  Inodes     IUsed    IFree      IUse%    Montado en
   /dev/sda1   14786560   704772  14081788   5%           /
   tmpfs       491569          5  491564     1%      /lib/init/rw
   udev        490359        625  489734     1%          /dev
```

Esta información es necesaria, ya que en el momento en que nos quedemos sin nodos-i, el sistema tendrá un desempeño más lento, por lo tanto es un parámetro a monitorear.

Ejemplo 2

```
# df -a

    S.ficheros  Bloques de 1K    Usado       Dispon     Uso%   Montado en
    /dev/sda1   232805336       103212532    117766936  47%         /
    Tmpfs       1966276                 0      1966276  0%     /lib/init/rw
    Proc        0                       0            0  -         /proc
    Udev        1961436               156      1966276  1%         /dev
    Devpts      0                       0            0  -        /dev/pts
    Sysfs       0                       0            0  -          /sys
```

Si con esa información verifica que el sistema de archivos está trabajando con espacio insuficiente, puede utilizar el comando ***du*.**

du	Se usa para estimar el uso de espacio en disco duro, de un archivo, un directorio en particular o de archivos en un sistema de archivos.

Este comando es la abreviatura de disk usage (uso de disco) es un comando estándar de los sistemas operativos de la familia Unix, y se utiliza para saber qué archivos ocupan más espacio en un sistema de archivos

Sintaxis:

```
du [opciones]
```

Este comando tiene una serie de indicadores (banderas), también llamados opciones, que se pueden utilizar para obtener información diversa.

-b [–bytes] Mostrar en bytes.
-s [–sumarize] Mostrar solamente el tamaño total de cada argumento.
-h [–human-readable] Imprime los tamaños de forma legible (e.g., 1K, 234M, 2G)

Ejemplo

Para conocer el resumen del uso del disco para un directorio con todos sus subdirectorios:

```
$ du -h /home
      0 K    /home/dan/.mozilla/extensions
      0 K    /home/dan/.mozilla/plug-ins
      0 K    /home/dan/.mozilla
      4.0 K  /home/dan/.cache
      4.0 K  /home/dan/.cache/abrt
      0 K    /home/dan/.config/abrt
      0 K    /home/dan/.config
      20 K   /home/dan
      20 K   /home
```

Cuota de Disco

Los sistemas Linux con gran cantidad de usuarios, tarde o temprano tienen el problema de usuarios que almacenan demasiada información en sus directorios de trabajo, pudiendo incluso en casos extremos llenar completamente discos duros haciendo el sistema inservible. Con el uso de cuotas de disco (disk quotas) es posible limitar la cantidad de espacio disponible para cada usuario o de manera global para todos.

Tipos de cuota

Por Bloques (blocks): Un bloque corresponde a 1kb y una cuota por bloques correspondería al total de bloques que un usuario puede utilizar en el sistema.
Por Inodos (inodes): Una cuota por i-nodos indicaría el total de i-nodos a los que el usuario tiene derecho.

Los límites pueden ser Duro o límite absoluto, en este caso el usuario no podrá excederlo y limite Suave que si puede ser excedido por el usuario, pero sería conveniente advertir cuando el límite de uso ya se ha excedido.

Para implementar, se debe decidir qué particiones dentro del archivo /etc/fstab se desean tener bajo límite de espacio (cuota). Las palabras clave son: *usrquota y grpquota;* estas dos palabras se especifican en el archivo /etc/fstab y se debe ser usuario root para modificarlo:

Ejemplo

```
# vi /etc/fstab
/dev/sda2  /          ext3    noatime          1       1
/dev/sda1  /boot      ext3    noatime          1       2
/dev/sda3  /home      ext3    noatime,usrquota,grpquota       1       2
                                  ...
```

Ahora bien, lo anterior deja listo el sistema de archivos /home para que soporte cuotas, pero no disponibles para su uso, es decir no están activadas. Para activarlas se hace con el comando *quotaon*.

quotaon/off	*S*e usa para activar/desactivar las cuotas definidas para un usuario o grupo es decir habilitar/deshabilitar su límite de espacio en disco.

Sintaxis:

```
quotaon [parámetros] sistema de archivos
```

Opciones:

- **-g** abrir límite de espacio de disco del grupo.
- **-u** abrir límite de espacio de disco del usuario.
- **-v** muestra instrucción de ejecución de comandos.

Ejemplo

```
# quotaon -ugv /home
/dev/sda3 [/home]: group quotas turned on
/dev/sda3 [/home]: user quotas turned on
```

Posteriormente hay que establecer la cuota por usuario, ya que por defecto ningún usuario las tiene establecidas. Para ello, se debe ejecutar el comando *edquota*.

edquota	es un editor que se usa para establecer o definir la cuota para un usuario.

Sintaxis:

```
edquota –u nombredeusuario
```

En `nombredeusuario` se puede especificar uno o más usuarios o grupos, también puede utilizarse el UID / GID.

Para cada usuario o grupo, se crea un archivo temporal de las cuotas de disco actuales y luego con el comando/editor *edquota* se pueden modificar las cuotas o agregar nuevas. Si se establece una cuota a cero, se indica que no se deben imponer cuotas.

El uso y los límites de los bloques se informan y se interpretan como múltiplos de bloques de kilobyte (1024 bytes) de forma predeterminada. Los símbolos K, M, G y T pueden agregarse a un valor numérico para expresar kilobytes, megabytes, gigabytes y terabytes.

Las columnas "blocks" e "inodes" son informativas, es decir nos indican la cantidad de bloques o inodos utilizados actualmente por el usuario, y las que podemos editar son las columnas "soft" y "hard" de cada caso.

Ejemplo

```
      # edquota –u user1
Disk quotas for user user1 (uid 502):
Filesystem            blocks       soft       hard     inodes     soft     hard
  /dev/sda3               56          0          0         14        0        0
```

El uso de inodos y los límites se interpretan literalmente. Los símbolos k, m, g y t se pueden agregar al valor numérico para expresar múltiplos de 10^3, 10^6, 10^9 y 10^{12} inodos.

Los usuarios pueden exceder sus límites flexibles por un período de gracia que puede especificarse por sistema de archivos. Una vez que el período de gracia ha expirado, el límite suave se aplica como un límite rígido.

La información de uso actual en el archivo es para fines informativos; solo los límites duros y blandos pueden ser cambiados.

Al salir del editor, edquota lee el archivo temporal y modifica los archivos de cuotas binarias para reflejar los cambios realizados.

El editor invocado es editor (1) a menos que el EDITOR o la variable de entorno VISUAL especifique lo contrario. Solo el superusuario puede editar cuotas.

Verificar el uso de las cuotas

Como usuario “root” se puede observar el uso de cuotas de cualquier usuario en forma individual con el comando:

quota	nos permite conocer el espacio utilizado por cada usuario o grupo.

Sintaxis:

```
quota –u nombreusuario
```

Ejemplo

```
# quota –u user1

Disk quotas for user user1 (uid 502):
Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
 /dev/sda3      56      70     100              14       0       0
```

Como usuario individual del sistema, puede observar su cuota con el mismo comando quota, sin argumentos. Ahora bien, si se desea un reporte global de las cuotas de todos los usuarios o por grupos, siendo "root" se utiliza el comando repquota.

Ejemplo

```
# repquota /home

*** Report for user quotas on device /dev/sda3
Block grace time: 7days; Inode grace time: 7days
Block limits                File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --  184280       0       0              11     0     0
sergio    -- 42579852       0 50000000          34902     0     0
user1     --       56      70     100              14     0     0
user2     --       52       0       0              13     0     0
user3     --       28       0       0               7     0     0
user4     --       28       0       0               7     0     0
```

Impresión en Linux

En sistemas en los que muchos usuarios comparten una o más impresoras, puede tener que esperar a que un trabajo se imprima. Cada impresora tiene una cola asociada, en la que se almacenan los trabajos hasta que esté disponible.

Una tarea básica de todo administrador de sistema entre otras, es la de imprimir archivos, examinar la cola de impresión, eliminar trabajos de la cola, formatear archivos antes de imprimirlos, y configurar el entorno de impresión.

Linux posee cinco programas fundamentales para facilitar la tarea del administrador y permitir el manejo de la impresión, estos archivos son: ***lpr, lpq, lpc*** y ***lprm***, que tienen por objetivo enviar, examinar y cancelar los trabajos de impresión, y se encuentran en el subdirectosio /usr/bin/ y el quinto programa llamado ***lpd*** que es el demonio de impresión y se encuentra en el sudirectorio /usr/sbin/**.**

Los programas, lpr, lprm, lpc y lpq trabajan por defecto con una impresora llamada "lp", pero si se define la variable de entorno PRINTER con el nombre de una impresora, pasará a ocupar el valor por defecto ocupado por lp. De la siguiente forma:

$ PRINTER="nombre_de_impresora"; export PRINTER

Como usuario root, la forma más simple de imprimir en Linux es enviar el archivo a ser impreso directamente al dispositivo de impresión.

Ejemplo

```
# cat tesis.txt > /dev/lp0
```

Pero para seguridad, sólo el usuario root y los usuarios de su mismo grupo, como el demonio de impresión, son capaces de escribir directamente a la impresora.

Por esto, los usuarios comunes tienen que usar *lpr* para imprimir un archivo sin esperar a que el trabajo que se está imprimiendo termine.

lpr comando que se utiliza para enviar un archivo al dispositivo de impresión

El comando lpr es responsable de preocuparse por el trabajo inicial para imprimir un archivo, pasando entonces el control a otro programa, ***lpd***, el demonio de las impresoras en línea. Este demonio le dice entonces a la impresora cómo imprimir el archivo.

Sintaxis:

```
lpr [ opciones ] [ nombre_archivo ... ]
```

El comando lpr copia el archivo a imprimir al directorio de spool, donde el archivo permanece hasta que el demonio lpd lo imprime. Cuando lpd se entera que hay un archivo para imprimir, creará un proceso 'fork'. Este nuevo proceso (copia de lpd), imprime el archivo mientras la copia original queda esperando otras peticiones, lo que permite múltiples trabajos de impresión en una cola.

Opciones

- **- Pprinter** especifica la impresora a usar a través de printer
- **- h** suprime la impresión de la página banner,
- **- # num** especifica el número de copias a imprimir
- **- s** crea un enlace simbólico en lugar de copiar el archivo completo al directorio de spooling (útil para archivos grandes)

Ejemplo

```
$ lpr -#2 -s -h -Pimpr  informe2.txt
```

Este comando define que se use la impresora llamada `impr` y crea un enlace simbólico al archivo `informe2.txt` en el directorio de spool, donde debería ser procesado por lpd e imprimirlo sin página banner y por dos copias.

Si no se especifica un nombre de archivo, lpr asume que la entrada será efectuada por la entrada estándar, es posible que el usuario redirija la salida de un programa al dispositivo de impresión.

Ejemplo

```
$ ls -l /bin | lpr -Pprinter
```

lprm este comando elimina un trabajo o archivo de la cola de impresión.

En caso de que se desee borrar un trabajo de la cola de impresión se deberá utilizar el comando ***lprm,*** de la siguiente forma:

Sintaxis

```
lprm [ - Pprinter] [ - ] [ job # ]  user
```

Opciones

- **- Pprinter** especifica la cola asociada a esa impresora
- **-** borra todos lo trabajos de un usuario en particular
- **user** especifica el nombre del usuario dueño del trabajo a eliminar
- **job #** hace referencia al número de trabajo a eliminar de la cola.

lpq	nos permite conocer qué trabajos están en la cola de impresión

Este comando nos facilita saber qué trabajos están actualmente en una cola de impresión particular.

Sintaxis:

```
lpq impresora
```

Ejemplo

Para mostrar la cola de la impresora por defecto (definida por /etc/printcap), solo se debe tipear:

```
$ lpq
Waiting for lp to becom ready (offline?)
Rank          owner       Job   Files              Total Size
1 root              0           informe2.txt      72048 bytes
2 root              1           listado          218248 bytes
3 luis              1      resumen                13230 bytes
4 ana               1      nota                    8230 bytes
5 mario             1      resultados              5390 bytes
```

La salida mostrada nos indica que los trabajos 0 y 1 asociados a los archivos *informe2.txt y listad*o respectivamente, están en estado de espera. Al mismo tiempo lpq nos informa que la impresora no se encuentra activada para realizar los distintos trabajos de impresión.

lpc	nos permite activar o desactivar la impresora.

Si lo desea puede activar la impresora introduciendo el comando ***lpc*** de la siguiente forma:

```
# lpc start all
      lp :
          printing enabled
          daemon started
```

Como ya vimos, con **lpc** podemos comprobar el estado de las impresoras, y controlar algunos aspectos de su uso. Particularmente, permite activar y desactivar la cola de impresión, permite activar y desactivar impresoras, y reorganizar el orden de los trabajos en cola. Con los comandos anteriores activamos todas las impresoras.

Ejemplo 1

Podemos desactivarla de la siguiente manera:

```
#  lpc down lp
```

Ejemplo 2

Tambien es posible mover el orden de los trabajos que se encuentran en la cola.

```
# lpc topq lp 5
```

Este comando mueve el trabajo que está en orden 5, que corresponde al usuario mario, y lo manda al principio de la cola.

Si no especificamos argumentos, lpc entrará en modo diálogo. Con "?" obtenemos ayuda. Advierta que la mayoría de las funciones de lpc están reservadas para el usuario root.

Nota: Recuerde que el comando man (manual) explica con detalle todas estas órdenes y debería consultar si desea ampliar información.

Directorios y archivos importantes

- El directorio de la cola de impresión, o directorio de spool, es donde se almacenan los datos a la espera de que **lpd** decida qué hacer con ellos y por tal motivo es un directorio muy importante, pero en un sistema común cada impresora debería tener su directorio de spool, lo que facilita notablemente el mantenimiento. En la mayoría de las instalaciones, /var/spool/lpd es el directorio de spool raíz, y cada impresora según su nombre debería tener un subdirectorio particular. Por ejemplo /var/spool/lpd/HP690.

- El archivo /etc/printcap es un archivo de texto, en donde cada entrada del archivo /etc/printcap describe una impresora. Es decir que cada entrada de printcap provee una denominación lógica para un dispositivo físico, y tambien definirá qué puerto vamos a usar, qué directorio de spool, qué proceso deben soportar los datos, qué clase de errores debemos notificar, qué volumen de datos se permiten enviar como máximo, o limitar el acceso de ciertos dispositivos, entre otros. Su propietario debe ser root y debe tener los permisos, -rw-r--r--.

Actividad 1

1. Inicie su sesión.

2. Visualice el archivo /proc/devices y analice su información.
   ```
   cat /proc/devices
   ```

3. Visualice el archivo /proc/pci y analice su información.
   ```
   cat /proc/pci
   ```

4. Muestre por pantalla el archivo /proc/ide y analice su información.
   ```
   more -l  /proc/ide
   ```

5. Visualice el archivo /proc/scsi y analice su información.
   ```
   more  /proc/scsi
   ```

6. Muestre el archivo /etc/mtab y analice su información, ¿qué comando utiliza esta información?
   ```
   cat /etc/mtab
   mount
   ```

7. Visualice el archivo /etc/fstab y analice su información.
   ```
   more /etc/fstab
   ```

8. Envíe a la cola de impresión los archivos /etc/passwd y /dev/mtab
   ```
   lpr -h  /etc/passwd
   lpr   /etc/mtab
   ```

9. Muestre la cola de impresión y analice la información
   ```
   lpq
   ```

10. Elimine de la cola de impresión los trabajos enviados por Ud.
    ```
    lprm  -user alumnoxy
    ```

Actividad 2

1. Inicie su sesión y visualice el archivo que contiene información sobre los dispositivos reconocidos por el kernel de Linux

2. El comando *mount* utliza información del archivo:
   a. /etc/mtab
   b. /dev/proc
   c. /etc/fstab
   d. /etc/passwd

3. Montar el sistema de archivos de un CD de Linux, y mostrar su contenido.

4. Observe cuánta cuota de disco está usted utilizando como usuario individual

5. ¿Cómo observo el uso del espacio de un usuario en particular?

6. ¿Puedo verificar el uso de las cuotas de todos los usuarios? ¿Cómo?

Actividad 3

1. ¿Qué comando usaría para mandar a cola impresión un archivo de texto?

2. ¿Qué diferencia tiene el comando **lprm y lpq**?

3. ¿Cómo reviso la cola de impresión?

4. ¿Qué comando se utiliza para chequear el estado de la impresora?

5. ¿Cómo elimino uno de los trabajos de la cola de impresión?

6. Explique qué es el punto de montaje.

7. Cuando se realiza el montaje de un dispositivo es necesario especificar el formato del sistema de archivos. ¿De qué se trata?

Actividad 4

1. ¿Cómo puedo ver todas las particiones y discos existentes?

2. ¿Cómo puedo ver el porcentaje de uso de un sistema recién montado?

a) Crear un dispositivo llamado DiscoX de 100 Mb con bloques de 512bytes, crearlo con ceros.

3. Mostrar si se creó con el tamaño correcto.

4. Particionar el dispositivo en dos particiones primarias de 20 MB c/u y una extendida de 60 MB con 3 particiones lógicas de tamaño diferente.

5. Mostrar el resultado del particionado.

a) Crear otro dispositivo llamado DiscoW de 200 Mb con bloques de 1Kb y con ceros.

b) Crear un sistema de archivo en él de tipo Linux ext.

6. Crear un punto de montaje (directorio) y montar el dispositivo.

7. Ver el uso del sistema de archivo recién montado

Capítulo 11

Administración de Usuarios y Grupos

Usuarios y grupos introducción

La gestión de usuarios y grupos en Linux, permite el manejo de múltiples usuarios, teniendo en cuenta que el usuario puede ser una cuenta de una persona, o una cuenta genérica de uso común por varias personas o cuentas que son usadas por las aplicaciones, siendo un componente clave del sistema operativo. Una de las consecuencias de esta administración es que ninguna de las tareas que se realicen por cualquier usuario, pueda poner en riesgo todo el sistema.

Los Grupos representan normalmente lógicas de organización donde se asocian usuarios que tienen funcionalidades en común. Los usuarios que pertenecen al mismo grupo pueden leer, escribir o ejecutar archivos que están asociados a ese grupo.

La gestión de usuarios y grupos deben acompañar las políticas de seguridad que se establecen para proteger los datos de los usuarios y acompañar la consistencia del sistema.

Cada usuario y grupo posee asociado un número de identificación único llamado identificador de usuario (UID) y un identificador de grupo (GID) respectivamente. De modo general cuando se crea un grupo el sistema operativo asigna un GID de 1000 o superior. Aquellos grupos que poseen un GID menor a 100 se consideran reservados y son de uso de grupos especiales o del propio sistema operativo.

Es el root quien puede realizar el cambio de propietario de un archivo, pero los permisos de acceso pueden ser modificados por el root o por el creador del archivo.

El administrador del sistema tiene una de las tareas más importantes en la administración correcta de usuarios y grupos, así como en la asignación y revocación de los permisos.

Archivos relacionados con la administración de usuarios y grupos

/etc/passwd contiene la información acerca de los usuarios

Cada línea de este archivo contiene información acerca de un único usuario; el formato de cada línea es:

Nombre de usuario : x : UID : GID : descripción : dir.inicio : intérprete

Ejemplo

```
Fabian : x : 102 : 100 : Fabian Guber : /home/fabian : /bin/bash
```

Nombre de usuario	Es el nombre de usuario, el identificador único dado a cada usuario del sistema.
X	Representa la contraseña cifrada: Indica que se hace uso de /etc/shadow, donde se encuentra la clave cifrada.
UID	Es un número único dado a cada usuario del sistema. El sistema normalmente mantiene la información de los permisos por UID, no por nombre de usuario.
GID	Es el número único que identifica al grupo.

Descripción	Se describe el nombre real del usuario o puede haber alguna descripción, también puede estar vacío.
dir.inicio	Es el directorio en el que se coloca inicialmente al usuario en el momento de conexión. Cada usuario debe tener su propio directorio inicial.
Intérprete	Es el intérprete de comandos que es arrancado para el usuario en tiempo de conexión.

Ejemplo

```
# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
…
redes:x:1000:1000:redes,,,:/home/redes:/bin/bash
lightdm:x:115:122:Light Display Manager:/var/lib/lightdm:/bin/false
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
vde2-net:x:105:116::/var/run/vde2:/bin/false
fabian:x:1001:1001::/home/fabian:/bin/sh
```

/etc/shadow contiene contraseñas cifradas

Cada línea de este archivo contiene información acerca de un único usuario; el formato de cada línea es:

Nombre de usuario : clave : UC : PC : DC : CC : AD : CD : R

Nombre de usuario	Es el nombre de usuario, el identificador único dado a cada usuario del sistema. Como está registrado en /etc/passwd.
clave	La contraseña se encuentra cifrada.
UC	Cantidad de días desde la fecha 01/01/1970 hasta el último cambio de contraseña.
PC	Mínimo de días que deben pasar hasta que se pueda cambiar la contraseña.
DC	Máximo de días que deben pasar para la caducidad de la contraseña y deba ser cambiada.
CC	Cantidad de días de antelación para que el sistema notifique la caducidad de la contraseña.
AD	Cantidad de días con la contraseña caducada antes que la cuenta se deshabilite.
CD	Cantidad de días desde la fecha 01/01/1970 y el día en que la cuenta se deshabilitó.
R	Es un campo reservado.

Ejemplo

```
# cat /etc/shadow
root:$6$eLy3tqjj$0k5SRyfxN.HdPeStbTndIY67qbzp7BMNe6ZJQk5uEVdPrL9UX2cNipWrhM
Y3zTpWm1GPOr3TJMQQCrBEOZVwU1:16392:0:99999:7:::
daemon:*:16392:0:99999:7:::
bin:*:16392:0:99999:7:::
sys:*:16392:0:99999:7:::
…
redes:$6$yisfLJ6P$zRb1kXVYh9HWOz9q9oh7VRTq3Y7LCCSP8qLmBTMtEB6397Whyh9mcgEWT
X94f1uBEvLHtzLqvkNMtBfSC40OK0:16392:0:99999:7:::
telnetd:*:16392:0:99999:7:::
```

```
uml-net:*:16392:0:99999:7:::
lightdm:*:16392:0:99999:7:::
vboxadd:!:17485::::::
vde2-net:*:17866:0:99999:7:::
fabian:$6$8BSLHV1z$VWk1mF1af.eWb89EmA3pSGWA1rH/.xg4ddBouIjHKk8eFkcwvXiXSY8a
4tNbFKz2gj0fvIBY74pMRP/lXDK/j/:17949:0:99999:7:::
```

/etc/group contiene información acerca de los grupos

Cada línea de este archivo contiene información acerca de un único usuario; el formato de cada línea es:

nombre de grupo : clave : GID : otros miembros

nombre de grupo	Es el nombre del grupo definido.
Clave	La contraseña cifrada para el grupo o una x indicando la existencia del archivo /etc/gshadow.
GID	Número de identificación del grupo.
Otros miembros	Lista de usuarios que pertenecen al grupo, esta lista debe estar separada por comas. También se pueden añadir usuarios que pertenecen a otros grupos, de este modo asumen los privilegios de ese grupo.

Ejemplo

```
# cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:
tty:x:5:
scanner:x:111:saned,redes….
lightdm:x:122:
wireshark:x:110:redes
vde2-net:x:116:
fabian:x:1001:
```

/etc/gshadow contiene contraseñas cifradas para grupos

Cada línea de este archivo contiene información acerca de un único usuario; el formato de cada línea es:

nombre : contraseña:uid:gid:descripción opcional carpeta:Shell

Nombre	Nombre de usuario
Contraseña	Si existe una x significa que la contraseña está cifrada en /etc/shadow. Si tenemos la x, los usuarios que no pertenezcan al grupo pueden unirse a éste escribiendo la contraseña para ese grupo usando el comando newgrp. Si el valor de este campo es un signo !, significa que ningún usuario tiene acceso al grupo usando el comando newgrp. Si el valor en nulo, solamente los miembros del grupo pueden conectarse al grupo.

UID	Número de identificador de usuario.
GID	Número identificador de grupo.
Carpeta	Directorio de inicio de sesión del usuario.
Shell	Existen usuarios que no tienen Shell.

Ejemplo

```
# cat /etc/gshadow
root:*::
daemon:*::
bin:*::
sys:*::
adm:*::
tty:*::
….
redes:!::
telnetd:!::
uml-net:!::redes
vboxsf:!::redes
lightdm:!::
wireshark:!::
vde2-net:!::
fabian:!::
```

Administración de Grupos

Podemos definir que cada usuario del sistema puede ser miembro de uno o más grupos. Hay grupos predefinidos en el sistema (bin, sys, daemon,…), donde los usuarios no pueden ser miembros, sino que son de uso exclusivo para el sistema. Los usuarios son miembros de grupos, por empezar se crea un grupo con su mismo nombre de usuario y se lo puede incorporar a otros grupos a través de comandos. Para realizar las tareas de administración de usuarios se debe estar logueado como usuario root o ejecutar cada línea de comandos anteponiendo la palabra reservada sudo.

Los archivos mencionados anteriormente son manipulados a través de los siguientes comandos:

groupadd crea una nueva cuenta de grupo

Sintaxis:

```
groupadd  [opciones] nombre-de-grupo
```

Opciones:

-g gid valor numérico del identificador de grupo. Este valor debe ser único. Los valores entre 0 y 99 se reservan normalmente para cuentas del sistema.

Ejemplo

```
groupadd  -g 520 alumnos
```

La creación de los grupos las puede verificar en el archivo /etc/group.

groupmod permite modificar atributos de un grupo ya existente

Sintaxis:

```
groupmod  [opciones] nombre de grupo
```

Opciones:

-g gid El valor numérico del identificador de grupo.
-n nombre-nuevo Cambia el nombre de la cuenta de grupo.

Ejemplo

```
groupmod -g 525 -n profesores  alumnos
```

En este ejemplo el comando cambia el gid del grupo alumnos de 520 a 525 y cambia el nombre, de alumnos a profesores.

groupdel elimina una cuenta de grupo

Sintaxis:

```
groupdel  nombre-de-grupo
```

Ejemplo

```
groupdel  docentes
```

gpasswd permite cambiar la contraseña del grupo

Sintaxis:

```
gpasswd  nombre-de-grupo
```

Ejemplo

```
# gpasswd docentes
Cambiando la contraseña para el grupo testx
Nueva contraseña:
Vuelva a introducir la nueva contraseña:
#
```

grpconv y grpuncov estos comandos convierten al formato gshadow o eliminan el formato gshadow

En la mayoría de los GNU/Linux modernos la protección extendida del archivo /etc/gshadow está activada.

```
# grpconv
```

Al ejecutar el comando se sincroniza el archivo group con gshadow y en este último queda registrado el hash de la clave de los grupos.

```
# grpunconv
```

Al ejecutar el comando se rompe la sincronización del archivo group con gshadow y este último se elimina, el hash de la clave de los grupos queda en group.

newgrp	permite hacer que un usuario cambie su grupo principal de forma temporal

Sintaxis:

```
newgrp  nombre-de-grupo-nuevo
```

Ejemplo

```
#newgrp docentes
```

Si el grupo nuevo está configurado con contraseña, la misma será solicitada al ejecutar el comando. Es decir, que los comandos que se ejecuten a posterior, lo harán bajo el nuevo grupo de trabajo. Para regresar y trabajar en el grupo inicial, puede ejecutar el comando exit.

Agregar, modificar y borrar usuarios

Las herramientas o comandos que se pueden utilizar para añadir, modificar y eliminar cuentas de usuarios son: *adduser, moduser* y *deluser* (se puede encontrar también como *useradd, usermod y userdel.)*

Para la creación de un usuario se dependerá del usuario con el que se realizó el inicio de sesión:

- Si se inició sesión con root el comando para crear un usuario **adduser o useradd.**
- Si no se inició sesión con root será necesario anteponer **sudo**. (sudo brinda privilegios administrativos al usuario que lo ejecuta).

useradd	crea una cuenta de usuario. Sin opciones toma valores por defecto

Pero podemos especificar los atributos de la cuenta, tal como: identificador de usuario, grupo al que pertenece, el directorio de conexión, etc, utilizando opciones.

Sintaxis:

```
useradd  [opciones]  nombre de usuario
```

Opciones:

-u uid	Número que identifica al usuario (**Unique Identification Number**)
-g grupo primario	Identificador de grupo (**Group Identification Number**)
-G grupo, ...	Lista de nombres de grupos secundarios
-d directorio	Ruta de acceso absoluta al directorio de conexión (inicio de sesión)
-c "comentario"	Texto con alguna observación
-s Shell	Ruta de acceso absoluta al intérprete de comandos
-e fecha de expiración	Fecha de caducidad de la nueva cuenta
-p contraseña	Clave de acceso

Ejemplo

```
 # useradd –u 510 –g ventas–G facturación logistica –d /home/mostrador/usu755
–s /bin/bash usu755
 # passwd usu755
```

En ese ejemplo creamos el usuario usu755, y le asignamos las siguientes características:

Grupo primario: ventas
Grupos adicionales o secundarios: facturación y logística
Directorio de inicio de sesión: /home/mostrador/usu755
Shell por defecto: /shell/bash

passwd cambia la contraseña del usuario

Sintaxis:

```
passwd [opciones] [USUARIO]
```

Opciones:

-e, --expire fuerza a que la contraseña de la cuenta caduque

Ejemplo

```
 # passwd usu755
```

usermod cambia los atributos del usuario una vez creado

Sintaxis:

```
usermod  [opciones] nombre de usuario
```

Opciones:

-c (–comment)	Establece el nombre completo del usuario y otros datos relevantes de la cuenta. También existe el comando chfn y sus opciones, para estas funcionalidades.
-d (–home)	Asigna un nuevo directorio HOME al usuario

-e (–expiredate)	Establece una fecha de expiración de la cuenta. El formato es YYYY-MM-DD Si se introduce una fecha de expiración vacía, se deshabilitará la expiración de la cuenta
-f (–inactive)	Establece en días, el tiempo en que se establecerá la clave del usuario como inactiva una vez alcanzada la fecha de expiración. El valor 0 establece que en cuanto llega la fecha de expiración, se establece como inactiva la clave del usuario y un valor -1 inhabilita esta opción
-g (–gid)	Cambia el grupo del usuario. Se debe usar nombre o número del grupo. Debe existir el grupo previamente
-G (–groups)	Establece los grupos a los que el usuario pertenecerá. Si el usuario ya pertenece a algún grupo del sistema y no se nombra este grupo al usar esta opción, desaparecerá de ese grupo. Para evitar ésto, se usa la opción -a (–append) que adjunta los grupos que hemos añadido al usuario a los que ya había creados. Los grupos se deben separar por comas y sin espacios
-L (–lock)	Bloquea la cuenta del usuario. Simplemente establece un '!' delante de la clave encriptada
-m (–move-home)	Mueve el contenido del HOME del usuario a una nueva localización
-o (–non-unique)	Sirve para establecer una ID no única al usuario
-p (–password)	Sirve para modificar la clave encriptada del usuario
-s (–shell)	Selecciona el nuevo shell que va a utilizar el usuario. Si se deja en blanco usará el establecido por defecto en el sistema
-u (–uid)	Establece el nuevo valor ID del usuario. Este valor debe ser numérico
-U (–unlock)	Desbloquea la cuenta del usuario. Quita el '!' delante de la clave encriptada

Ejemplo

```
# usermod   -d   /home/atcliente/vendedor1  -m  -s /bin/sh   usu755
```

En este caso hemos modificado el directorio de conexión y el shell del usuario creado anteriormente. Con la opción m, mueve el contenido del directorio de conexión anterior al actual, es decir a */home/atcliente/vendedor1*

```
#usermod  -L usu755
```

Si por alguna razón el usuario usu755 no puede ingresar al sistema, podemos deshabilitar la cuenta sin tener que borrarla. Lo hacemos con la opción –L del comando usermod. El comando este caso modifica el archivo /etc/shadow y antepone el signo **'!'** a la clave encriptada. Es posible obtener el mismo efecto si utilizamos un editor de texto y le agregamos un asterisco ("*") delante del campo de la clave en /etc/passwd. En tal caso podemos ver los datos de la cuenta del usuario usu755, como:

*usu755:*x:102:100:atención al cliente:/home/atcliente/vendedor1:/bin/bash*

userdel elimina la cuenta de usuario

Sintaxis:

```
userdel  nombre-usuario
```

Ejemplo

```
# userdel  usuarioxxx
```

pwconv y pwuncov	estos comandos convierten al formato shadow o elimina el formato shadow

En la mayoría de los GNU/Linux modernos la protección extendida del archivo /etc/shadow está activada.

```
# pwconv
```

Al ejecutar el comando se sincroniza el archivo passwd con shadow y en este último queda registrado el hash de la clave de los usuarios.

```
# pwunconv
```

Al ejecutar el comando se rompe la sincronización del archivo passwd con shadow y este último se elimina, el hash de la clave de los usuarios queda en passwd.

Otros comandos para usuarios y grupos

chown	permite cambiar el propietario de un archivo o directorio. El usuario root puede cambiar el propietario de cualquier archivo o directorio. En ocasiones, por razones de seguridad, está restringida esta operación para los usuarios normales

Sintaxis:

```
chown [-opciones] nuevo-propietario archivos/directorios
```

Opciones:

- **-R** Cambia el permiso en archivos que estén en subdirectorios del directorio en el que esté posicionado en ese momento.
- **-c** Muestra un mensaje donde menciona solamente aquellos archivos cuyo propietario cambia realmente.

Si se indica más de un archivo, van separados por espacio. Puede especificarse, también, en vez del nombre de un usuario, el identificador de usuario (UID) y el identificador de grupo (GID). Opcionalmente, utilizando un signo de dos puntos (:), o bien un punto (.), sin espacios entre ellos, entonces se cambia el usuario y grupo al que pertenece cada archivo.

Ejemplo

Modifica de un archivo el propietario y el grupo

```
# chown root:root testx/archivo1.txt
```

Se utiliza para cambiar el propietario de un archivo (o directorio), solo el dueño del archivo puede otorgar el archivo a otro usuario. Pero recuerde que *root* es el copropietario (invisible) de todos

los archivos en el sistema, por lo que él puede ejecutar *chown* para cambiar el propietario de cualquier archivo.

chgrp permite cambiar el grupo de usuarios de un archivo o directorio

Sintaxis:
```
chgrp nuevo_grupo archivo1 [archivo2 archivo3...]
```

En el caso de un directorio se cambia solo el dueño del directorio y no del contenido de este. Para realizar el cambio de grupo de todo el directorio con su contenido la sintaxis del comando es:

Ejemplo

```
# chgrp  -R  nuevo_grupo  directorio
```

chsh permite cambiar el shell del usuario activo

Sintaxis:
```
chsh  ruta_absoluta_del_nuevo_shell
```

Ejemplo

```
 # chsh /bin/ash
```

who lista los nombres de los usuarios conectados actualmente al sistema. Se visualiza también el tiempo que han estado conectados y el nombre del host desde el que se han conectado.

La ejecución de este comando nos devuelve el nombre de usuario de cada uno de los usuarios que ha iniciado una sesión en el sistema, junto con el terminal, la fecha y hora de inicio y las direcciones IP desde las cuales están conectados.

Sintaxis:
```
who [-opciones]
```

Opciones:

-b Muestra la hora del último arranque del sistema
-d Muestra los procesos muertos
-H Muestra los encabezados de columna encima de la salida

Ejemplo

```
$ who
Fabian      tty01       4:23    Mar 28  5:38
Monica      tty02       3:50    Mar 28  6:10
```

/etc/login.defs	en este archivo se definen las variables que permiten controlar los valores por defecto al crear un usuario y los valores por defecto del archivo /etc/shadow

Campos del archivo:

MAIL_DIR	Directorio en el que se guarda la cola de correos electrónicos de salida
PASS_MAX_DAYS	Número máximo de días que una contraseña es válida
PASS_MIN_DAYS	Número mínimo de días que una contraseña es válida
PASS_MIN_LEN	Número mínimo de caracteres en la contraseña
PASS_WARN_AGE	Número de días antes de que la contraseña expire, el sistema mostrará un mensaje de advertencia
UID_MIN y UID_MAX	Rango de valores de ID para ingresar para los usuarios
GID_MIN y GID_MAX	Rango de valores de ID para ingresar para los grupos
UMASK	Máscara de permisos. Si no se especifica, el permiso se define en 077
CREATE_HOME	Indica si un directorio de inicio debe crearse de forma predeterminada para nuevos usuarios

Actividad 1

1. Inicie una sesión como superusuario o administrador del sistema
Login: root
Password:

2. Muestre el archivo /etc/passwd y analice su información.
cat /etc/passwd

3. Muestre el archivo /etc/shadow y analice su información.
cat /etc/shadow

4. Cree el grupo Alumnos y los usuarios alumnoA, alumnoB y AlumnoC que pertenezcan al grupo creado. Verifique en el archivo que corresponda que los nuevos usuarios y grupos existan.
addgroup Alumnos
adduser -G Alumnos alumnoA
adduser -G Alumnos alumnoB
adduser -G Alumnos alumnoC
cat /etc/passwd
cat /etc/group

5. Modifique el shell que utilizará el usuario alumnoB.
usermod -s /bin/ash alumnoB

6. Si quisiera cambiar el shell del usuario que está activo en este momento ¿qué comando usaría?
chsh /bin/ash

7. Cambie el grupo al que pertenece el subdirectorio copia y todos los archivos que contiene. Cambiarlo al grupo Alumnos.
chgr –R Alumnos ./copia

8. Muestre por pantalla los grupos a los que pertenece el usuarioA.
groups usuarioA

9. Borre el usuario alumnoB y alumnoC
userdel alumnoB
userdel alumnoC

10. Abra una sesión de trabajo como un usuario común.
Login: usuarioxx
Password:

11. ¿Quién es el propietario del archivo /etc/passwd? ¿Y el grupo? ¿Cuál es la lista de permisos?
_ _
ls – l /etc/passwd

12. Cree en su directorio de login un subdirectorio llamado "copia" y copie dentro de éste el archivo /etc/passwd. ¿Quién es ahora el propietario del archivo copiado? ¿Los permisos son iguales?
_ _
mkdir copia
cp /etc/passwd ./copia
ls –l ./copia/passwd

13. Cambie el propietario del archivo *passwd* (que como ya vio, usted es el dueño y está en el subdirectorio copia) para que pertenezca al usuario alumnoC. Verificar el cambio.

chown alumnoC ./copia/passwd
ls –l

14. Vuelva a cambiar el propietario del archivo anterior para dejarlo como antes, es decir que el propietario sea usted.

chown usuarioxxx ./copia/passwd

Como verá no es posible ejecutar este comando ya que usted ya no es el propietario del archivo y no tiene permisos sobre él.

Actividad 2

1. Inicie su sesión como supervisor o administrador.

Login: root
Password:

2. Muestre el archivo /etc/group y analice su información.

cat /etc/group

3. ¿Quién es el propietario del archivo /dev/hda1? ¿A qué grupo pertenece?

_ _

ls –l

4. Muestre los permisos que posee el archivo y explique.

ls –l

5. Copie el archivo anterior (/dev/hda1) en su directorio de login. ¿Quién es el propietario ahora? ¿Los permisos son los mismos? ¿Por qué?

cp /dev/hda1 .
ls –l hda1

6. Cree el grupo Docentes y los usuarios docente1, docente2 y docente3 que pertenezcan al grupo creado. Verifique en el archivo que corresponda que los nuevos usuarios y grupos existan.

addgroup Docentes
adduser -G Docentes docente1
adduser -G Docentes docente2
adduser -G Docentes docente3
cat /etc/passwd
cat /etc/group

7. Abrir en otra terminal una sesión del docente1 y crear el archivo *datos* con sus datos personales.

Login: docente1
Password:
cat > datos
nombre, apellido, domicilio, etc
Ctrl. d

8. Cambiar el propietario del archivo */home/docente1/datos*, de tal manera que el dueño sea el docente2. Verificar el cambio.

chown docente2 /home/docente1/datos
ls –l

9. Como supervisor modifique el shell que utilizará el usuario alumnoB.
usermod –s /bin/ash alumnoB

10. Muestre por pantalla los grupos a los que pertenece el docente2.
groups docente2

11. Borrar el usuario docente1 y docente2
userdel docente1
userdel docente2

12. Ejecute el comando *who* seguido de su nombre de usuario: ¿Cómo se visualiza la salida de dicho comando?

Capítulo 12

Comunicación de usuarios

Linux es un sistema operativo multiusuario, y obviamente comparte esta característica con todos los sistemas Unix modernos. Permite aprovechar al máximo esas capacidades y una de ellas es que se puede utilizar a Linux como servidor de aplicaciones; desde sus terminales, los usuarios pueden conectarse a través de una red con el servidor y ejecutar aplicaciones en él, se puede además compartir recursos como datos, periféricos y dispositivos de almacenamiento. Pero una característica a resaltar es que Unix fue creado en principio como un Sistema Operativo que facilitaría la comunicación entre computadoras y con otros Sistemas Operativos. De hecho una de las principales motivaciones de los autores de Unix, fue poder usar computadoras pequeñas para crear programas en distintos lenguajes de alto nivel, compilarlos, etc., y una vez depurados enviarlos a través de una conexión a otra computadora para ejecutarlos.

Linux, entonces, como sistema Unix, le permite al superusuario (root) avisar a los usuarios los cambios que se produzcan en el sistema, (funcionamiento, capacidad, nuevos servicios, servicios extinguidos, nuevos usuarios, nuevas normas, noticias, etc)

En todo sistema multiusuario que funcione como red se hace necesario que los usuarios se comuniquen entre sí, para ello existen comandos que facilitan esas funciones. Dentro de estos estudiaremos *wall, rwall, write, mesg y mail.*

Para solucionar esto existe un archivo /etc/motd el cual podrá ser editado por root, y exponer un comunicado que será presentado a todos los usuarios que se conecten a la máquina tras identificar su login y su clave de acceso.

wall envía un mensaje a <u>todos</u> los usuarios del sistema

Se debe tener en cuenta que este comando solo puede ser utilizado por el superusuario o root y solo lo recibirán los usuarios que tengan su sistema configurado para poder recibir estos mensajes. El mensaje también emitirá un pitido para atraer su atención.

Sintaxis:

```
wall  (enter)
 texto del mensaje
 Ctrol+D
```

<u>Ejemplo</u>

El administrador del sistema desea enviar un mensaje a todos los usuarios para que cierren su sesión. El comando sería el siguiente:

```
$ wall  (enter)
Por favor cerrar su sesión para mantenimiento del sistema.
Ctrol+D
```

Cada uno de los usuarios recibirá en pantalla lo siguiente:

```
Broadcast message from root (tty1)  Mon  May 20 15:45 2019 ….
Por favor cerrar su sesión para mantenimiento del sistema.
```

Luego de recibido el mensaje en cada una de las terminales, el cursor espera el ingreso de un comando. Ingresando Enter se regresa al shell.

rwall es similar a *wall*, salvo que envía un mensaje a todos los usuarios de la *red local* y no solo a los que están en su sistema.

Sintaxis:

```
rwall  (enter)
   texto del mensaje
   Ctrol+D
```

Como se ve la sintaxis es igual que *wall.*

Los sistemas deben ejecutar ***rwalld*** (el daemon de rwall) para recoger y emitir mensajes *rwall*

write este comando envía mensajes de texto a otros usuarios.

Sintaxis:

```
write  [ Usuario / Terminal ]
```

Donde se podrá utilizar una de las opciones o ambas (usuario / terminal). Luego de ingresar este comando por teclado, se podrá escribir el texto del mensaje que se desea enviar.

A medida que el texto se va justificando, es decir presionando Enter, el mensaje es enviado al terminal destino del mismo. Para finalizar deberá teclear Ctrl+d para volver al shell.

En la terminal del usuario destino aparece un mensaje indicando que existe otro usuario que se está comunicando con él, y podrá responder también ejecutando el comando *write*.

Siempre que el usuario tenga habilitada la recepción de mensajes estos aparecerán sin que el usuario receptor pueda evitarlos. La recepción de estos mensajes puede ser deshabilitada por el usuario utilizando el comando *mesg*.

mesg este comando habilita o deshabilita la comunicación entre usuarios por medio de *write*.

Si el usuario tiene el *mesg* en “no” o está realizando una tarea específica no podremos comunicarnos con *write*. Sin opción, muestra el estado actual del usuario (escribir o no).

Sintaxis:

```
mesg [n/y]
```

y permite que los usuarios del sistema le escriban mensajes.
n prohíbe que los usuarios del sistema le escriban mensajes.

En realidad este comando modifica los permisos al archivo específico del terminal donde está trabajando el usuario, por ejemplo, el archivo /dev/pts/10 tendrá deshabilitado el permiso de escritura para el grupo y los otros, si es que el usuario de esa terminal ha ejecutado el comando *mesg n*.

echo Permite la comunicación unidireccional con otro usuario que esté en el sistema en ese momento; básicamente realiza la misma función que write pero utilizando un mecanismo diferente.

Sintaxis:

```
echo "mensaje" > terminal
```

Ejemplo

```
$echo "Hola que tal" > /dev/ttyp3
```

talk se utiliza para la comunicación bidireccional entre dos usuarios que estén en el sistema en ese momento.

El comando *talk* avisa al segundo usuario que el primer usuario quiere hablar con él. Este comando indica a la otra persona que debe teclear para inicializar la sesión.

Sintaxis:

```
talk nombreusuario
```

mail el comando mail permite enviar correo a un usuario o una lista de usuarios.

Sintaxis:

```
mail Usuario1 Usuario2 Usuario3 ...
```

El comando ***mail*** admite la opción ***–s*** para especificar un tema o subject para el e-mail que se va a mandar. Esta opcion ***–s*** acepta como argumento la palabra que le siga, todo lo demás será considerado como nombre del usuario a enviar. Si el tema posee más de una palabra, el tema deberá ir entre comillas simples para indicarlo.

Ejemplo

```
$ mail  -s ´Reunión del martes´  ana pedro luis
```

Cuando se envía un mensaje a un usuario que tiene abierta una sesión, el sistema no informa la recepción del mensaje, sino que lo hace cuando este usuario inicia una nueva sesión, por lo que un usuario deberá chequear voluntariamente su buzón de mensajes.

El comando *mail* sin opciones ni argumentos visualiza el correo del usuario activo, mensaje a mensaje.

```
$ mail (enter)
```

La salida de este comando sería algo como lo que sigue:

```
"var/spool/mail/alumno100":  3 messages   2 news   3 unread
U1 alumno103@labsis.frc.utn.edu.ar friday Set 4 8:55    "Nos juntamos esta noche"
N1 docente25@labsis.frc.utn.edu.ar monday Set 7 10:40   "Práctico próxima clase"
N2 alumno115@labsis.frc.utn.edu.ar tuesday Set 8  4:26    "¿Como va todo?"
&
```

Nótese que al final de la salida de pantalla, aparece el símbolo **&.** Este símbolo es un indicador que la utilidad mail espera que ingresemos un subcomando. ***mail*** proporciona subcomandos para facilitar las operaciones de leer, guardar, borrar y responder a los mensajes. Entre ellos:

(Enter)	visualiza el próximo mensaje
d	borra el mensaje y visualiza el próximo
s archivo	guarda el mensaje en ***archivo***
w archivo	guarda el mensaje en ***archivo*** pero sin la cabecera.
m [usuarios]	envía el mensaje a los usuarios nombrados
q	vuelve al shell
x	vuelve al shel sin modificar el contenido del buzón
!comando	escapar al shell y ejecutar el comando
?	esta opción visualiza por pantalla un resumen de toda las opciones de mail

Cada usuario en Linux tiene un buzón en el que recibe el correo que le está dirigido. El e-mail se guarda en el buzón hasta que el usuario lo lee o lo borra. Una vez que se ha leído el e-mail, este se puede mover a un buzón secundario o personal. Este buzón secundario se llama *mbox* y se encuentra en el subdirectorio */home* del usuario, sin embargo el usuario puede indicar el nombre de un archivo como buzón secundario con la opción ***s.***

Es posible enviar mensajes a uno a varios usuarios de un mismo sistema, o de la red LAN a la que se encuentre conectado, y si está conectado a Internet puede mandar a cualquier parte del mundo.

Actividad 1

1. Verifique si tiene habilitada la opción de recibir mensajes.
```
mesg
```

2. Visualice los permisos del archivo pts que corresponda a su terminal.
```
who am i
ls – l /dev/pts?
```

3. Deshabilite la recepción de mensajes de su terminal y verifique el cambio realizado.
```
mesg n
ls – l /dev/pts?
```

4. Envíe un mensaje al usuario de la terminal más cercana a la suya.
```
write alumnoxy
```

5. Si el usuario usuario12 está conectado, inicie una comunicación con él, ¿puede realizar esta actividad con otro comando? Explique.
```
who | grep alumno12
write alumno12

otra opción a write es talk
```

6. Visualice su casilla de correo. Y analice los datos que se muestran.
```
mail
```

7. Envíe un correo electrónico al usuario alumnoxy+2 (xy es el número incluido en su nombre de usuario), cuyo tema sea Turno de exámen Dic/2019.
```
mail –s  "Turno de exámen Dic/2019"  alumnoxy+2
```

8. Usted es administrador de un sistema y necesita avisar a los usuarios que cierren todas las aplicaciones en 5 minutos.
```
wall
El sistema se cerrará en 5 minutos, por favor cierre su sesión.
```

9. Visualice las opciones de mail.
```
mail
& ?
```

10. Guardar los mensajes recibidos en un archivo de correo diferente a mbox.
```
mail
& s mensajes
```

11. Guardar los mensajes recibidos, sin cabecera, en un archivo de correo diferente a mbox.
```
mail
& w mensajes
```

Actividad 2

1. ¿Cómo funciona el comando ***talk*** y qué diferencia presenta con respecto a ***write***?

2. ¿Cómo funciona el comando ***echo*** aplicado a comunicación de usuarios?

3. ¿Cuál es la función del comando **wall**?

4. ¿Cuál es el contenido de la variable **mail**?

5. ¿La técnica de redireccionamiento en qué comando se utiliza?
6. Mencione los comandos en donde la comunicación es unidireccional y en cuales bidireccional.

Actividad 3

1. Envíe un mensaje al usuario Oliva. Utilice todos los comandos que conoce.

2. ¿Qué información nos muestra el archivo /dev/pts? ?

3. Si le envían un mensaje y no puede recibirlo, ¿cómo debe habilitar la recepción?

4. Guardar los mensajes recibidos en un archivo de correo llamado cartas.

5. Envíe un mensaje como administrador a todos los usuarios del sistema.

6. Envíe un correo al usuario Juárez cuyo tema sea Reunión próxima semana.

Capítulo 13

Programación del Shell

En este capítulo veremos que el shell puede aceptar no sólo órdenes simples, desde la línea de comandos, sino también secuencia de órdenes, grupo de órdenes y órdenes condicionales. Pero, además de ser el shell un intérprete de comandos, proporciona la posibilidad de almacenar líneas de comandos en un archivo para su posterior ejecución. Se le solicita al Shell la interpretación de un script donde se leerá cada una de las sentencias que el archivo tiene y se ejecutará cada una de ellas sin necesidad de que intervengamos nosotros como usuarios.

A este archivo de comandos se lo denomina **shellscript** y se lo puede crear con cualquier editor de textos, como puede ser vi, ed, nano o el editor del mc, entre otros. (El apéndice II trata el editor **vi)**.

El shell dispone de estructuras de programación que nos permiten definir la ejecución condicional de instrucciones, ejecución de programas con múltiples ramificaciones, bucles y agrupación en módulos de las instrucciones en subrutinas o funciones.

Hay que tener en cuenta que estamos en presencia de un lenguaje interpretado, que permite al programador realizar pruebas rápidas que no necesitan ser compiladas.

Para escribir programas en shell debemos primero conocer cómo definir y utilizar variables, luego conocer las estructuras de control que nos permiten controlar el flujo de ejecución de los scripts.

Shellscript	archivo de comandos que puede ejecutarse en el intérprete de comandos Shell.

Scripts ejecutables

Al ejecutar el script, se dispara un nuevo proceso que lo ejecuta en un subshell, una vez que terminan de ejecutarse las líneas del script, el subshell y el proceso terminan.

Según lo definido anteriormente podemos decir que hay ciertos comandos que no tienen impacto sobre el Shell que invoca el script, ya que estos comandos se ejecutan en el subshell. Por ejemplo si necesitamos realizar un script que cree un directorio y se cambie al mismo, como podemos observar en la Figura 1:

```
[sop@sistemas-~/SistOperativos]>cat > programa
pwd
mkdir directorio
cd directorio
pwd
^C
[sop@sistemas-~/SistOperativos]>
```

Figura1: Creación de un script

Cuando ejecutamos el script que hemos creado podremos observar que el mismo ejecuta todos los comandos, pero el comando de cambio de directorio no tiene efecto sobre el Shell donde se disparó el script, ya que el mismo es ejecutado en un proceso que se ejecuta en un subshell [Figura 2].

```
[sop@sistemas-~/SistOperativos]>chmod u+x programa
[sop@sistemas-~/SistOperativos]>
[sop@sistemas-~/SistOperativos]>./programa
/home/redes/SistOperativos
/home/redes/SistOperativos/directorio
[sop@sistemas-~/SistOperativos]>
```

Figura 2: Ejecución del script creado (Nivel de subshell)

Antes que podamos ejecutar nuestros *scripts* debemos disponer del permiso de lectura y ejecución para que el shell pueda leerlo y ejecutarlo. Para ello utilizamos el comando *chmod*.

Ejemplo

```
chmod u+x programa
```

Recordemos que de este modo le damos permiso de ejecución solo al usuario. Es decir agregarle el permiso de ejecución al archivo, y de esa manera poder llamarlo desde la línea de comandos simplemente como:

```
./programa
```

Con la cadena programa, hacemos referencia al nombre del script que se ha creado (en el ejemplo se llama programa). Al teclear el nombre del programa (shellscripts), el shell comprobará en primer lugar si dicho archivo es ejecutable. Si es así comenzará a ejecutarlo línea a línea (ejecución interpretada de comandos), ya que cada línea se interpreta inmediatamente antes de su ejecución, sin la necesidad de compilar el programa.

Los dos caracteres ubicados delante del nombre del scripts **./**, le indican al shell que el script es un ejecutable que se encuentra en el directorio actual. Por lo general, por motivos de seguridad, nuestro directorio no está incluido dentro de la variable PATH, y si no especificamos donde se encuentra el script, el shell no podrá ejecutarlo. Sin embargo, es posible agregar nuestro directorio a la variable PATH simplemente agregando la ruta a la variable PATH=**.:**$PATH.

Resulta conveniente crear un directorio bajo el directorio de conexión del usuario, denominado *bin*, con el fin de contener este tipo de archivos. Además deberemos modificar la variable PATH, para que incluya la trayectoria *$HOME/bin*

Continuando con el ejemplo del script que crea un directorio y se cambia al mismo, si se debe cambiar el directorio en el Shell desde donde se invoca y no en el subshell, es necesario utilizar el caracter **.** (punto) como muestra la Figura 3:

```
[sop@sistemas-~/SistOperativos]>. ./programa
/home/redes/SistOperativos
/home/redes/SistOperativos/directorio
[sop@sistemas-~/SistOperativos/directorio]>
```

Figura 3: Ejecución del script creado (Nivel de shell)

Variables del shell

Una variable es un objeto que tiene un valor determinado. Las variables son muy importantes ya que la mayoría de los programas las utilizan para obtener de ellas, valores de configuración.

Además una de las características de Linux es que nos permite escribir scripts o programas para el shell, por lo que se hace necesario conocer el uso de las variables. Cuando se crean las variables, el shell puede almacenarlas en dos zonas:

- la zona de datos locales
- la zona de datos de entorno.

Cuando se define una variable desde la línea de comandos, se almacena en la zona de dato local. Es una variable propia del shell actual. Cualquier variable que resida en la zona de datos local de un proceso, será ignorada por el/los procesos hijos.

Si el usuario cambia de shell o inicia otra sesión, las variables locales definidas anteriormente no serán reconocidas.

Es posible transferir una variable de la zona de datos local al entorno utilizando el comando *export*, acompañado del identificador de la variable que desea trasladar al entorno.

En la zona de entorno residirán todas las variables cuyos valores son reconocidos por todos los shell que ejecute un determinado usuario. Por ejemplo, durante el proceso de conexión, se definen algunas variables que se almacenan en la zona de datos de entorno, como puede ser la variable PS1, que guarda el carácter con el cual identificamos al shell.

Formas para crear una variable

Al crear una variable debemos pensar qué identificador o nombre vamos a utilizar para la misma. El identificador de la variable debe comenzar con un carácter alfabético, luego puede contener caracteres numéricos y el carácter de subrayado. No hay restricciones para la longitud del identificador o nombre de la variable. Las únicas variables del shell son del tipo carácter.

Para crear una variable podemos utilizar el método de *asignación*, *sustitución de comandos* o las órdenes *read* y *declare,* que detallaremos a continuación.

Asignación directa

Es posible asignarle un valor a una variable desde la línea de comandos, especificando el nombre de la variable, luego el signo "=" y, a continuación, el valor a asignar. No debe haber espacios a ambos lados del signo "=", para que el shell no trate a la asignación como si fuera una línea de comandos.

Ejemplo

```
$a=Hola
$b=Amigo
```

a y b son los nombres de las variables, "Hola" y "Amigo" es el contenido de cada una de ellas. Si necesitamos asignarle a una variable una cadena de caracteres, ésta deberá estar entre comillas simples o dobles para que el shell ignore los espacios en blanco, y no los tome como separador de campos.

Ejemplo

```
$a="Hola amigo"
```

Podríamos asignarle también a una variable la ruta de acceso a un archivo que utilizamos frecuentemente, y luego simplemente hacer referencia a la variable.

Ejemplo

```
$arch=/home/garcia/listado
```

Sustitución de comandos

Otra forma de asignar un valor a una variable, es utilizar la *sustitución de comandos*. Este procedimiento permite que una variable almacene la salida de un comando. La sintaxis para los shell Bourne y Korn (ya estudiados en el capítulo 1) consiste en utilizar las comillas graves.

Ejemplo

```
$fecha=`date`
```

El comando *date* se ejecuta y guarda su salida en la variable fecha. El shell POSIX requiere delimitar el comando a ejecutarse entre paréntesis y lo hace preceder del símbolo "$".

Ejemplo

```
$fecha=$(date)
```

El comando *date* se ejecuta y guarda su salida en la variable fecha.

Declaración de variables

declare	este comando permite declarar variables o modificar los atributos de las variables ya definidas.

Sintaxis:

```
declare  [opciones]  [var-nom[=valor]]
```

Las opciones son las siguientes:

-f Muestra sólo nombres de función.
-/+ i (-) Activa o (+) desactiva el atributo de la variable para que sea un entero.
- r (-) Activa el atributo de la variable y la define como de sólo lectura. En este caso no tiene efecto la opción +r
-/+ x (-) Activa o (+) desactiva el atributo de la variable para que pueda ser exportada al entorno.

El comando declare sin argumentos muestra las variables ya declaradas.

Ejemplo

```
$declare –r var1
```

El comando define a la variable como de sólo lectura.

Lectura de datos de la entrada estándar

read	este comando lee los datos de la entrada estándar, y asigna cada campo ingresado en la línea de comandos a una variable diferente, recuerde que el espacio es separador de campos.

Si el número de campos ingresados supera la cantidad de variables definidas, entonces, todos los campos sobrantes se asignan a la última variable definida.

Sintaxis:

```
read  var1 var2 ...
```

Ejemplo

```
$read dia mes año
```

Luego de ingresar el comando de la línea anterior, aparecerá el cursor esperando que usted ingrese el dato, por ejemplo, *28 Marzo 2019*. Al presionar la tecla <Enter>, se guarda la palabra 28 en la variable *dia*, la palabra Marzo en la variable *mes* y la palabra 2019 en la variable *año*.

Con *read*, no es posible redireccionar, por ejemplo un archivo a una variable pues este comando sólo lee la entrada estándar.

Uso del contenido de una variable

Es posible utilizar el contenido de una variable en una línea de comando simplemente haciéndola preceder del símbolo "$", procedimiento al que llamamos *sustitución de variables*.

Ejemplo

```
$cat $arch
```

En el ejemplo anterior, el shell, primero sustituye la variable arch por su valor y luego ejecuta el comando. El comando anterior muestra el contenido del archivo */home/garcia/listado*, valor que habíamos asignado anteriormente.

Si el identificador de la variable debe estar concatenado a una cadena de caracteres o bien a otra variable, se hace necesario encerrar al identificador entre llaves.

Ejemplo

```
$echo "Hoy es ${dia} de ${mes}"
```

La salida de esta línea de comandos es:

Hoy es 28 de Marzo

Existen otros métodos para utilizar el contenido de una variable. A continuación presentamos una tabla con algunos de los métodos disponibles para acceder a las variables.

$variable	Sustituye $variable por el valor de la variable.
${variable}	Sustituye ${variable} por el valor de la variable. Este método se emplea cuando existen otros caracteres inmediatamente a continuación del nombre de la variable y que serían interpretados como parte de dicho nombre.
${variable-palabra}	Sustituye ${variable-palabra} por el valor de la variable, si es que la variable está configurada. Sino se utiliza como valor, palabra.
${variable+palabra}	Sustituye ${variable+palabra} por el valor de la variable, si es que la variable está configurada. Sino se ignora la instrucción.
${variable=palabra}	Asigna el valor de palabra a variable, si variable no estuviera configurada.
${#variable}	Sustituye ${#variable} por la longitud, en caracteres, del valor de variable.

Tabla: Métodos alternativos de acceso a variables en bash [Thomas Scheck at al]

Argumentos posicionales y variables especiales

Mostramos en el siguiente cuadro una descripción resumida de los argumentos posicionales y las variables especiales utilizadas por el Shell. Cuando se dispara un script desde el Shell posterior a ingresar el mismo se pueden pasar argumentos, estos parámetros son accedidos mediante

variables. Las variables almacenan un valor relacionado a la ocurrencia de un evento en la ejecución de un comando.

Variable	Descripción
$?	Almacena el estado de salida del último comando ejecutado.
$!	Contiene el identificador del último proceso que se ejecutó en background.
$ -	Contiene las opciones establecidas mediante el comando *set.*
$#	Contiene el número de argumentos posicionales que fueron pasados al script.
$0	Nombre del shell script
$1,$2 ...$9	Argumentos posicionales
$* $@@	Lista con los argumentos posicionales.
$$	Contiene el identificador del proceso actual

Ejemplo

```
[sop@sistemas-~/SistOperativos]>cat -n VerArgumentos
     1  echo Nombre del script: $0
     2  echo Cantidad de argumentos: $#
     3  echo Todos los argumentos: $*
     4  echo primer argumento: $1
     5  echo segundo argumento: $2
     6  echo tercer argumento:$3
     7  shift #mover el argumento hacia la izquierda
     8  echo primer agumento despues de moverlos: $1
[sop@sistemas-~/SistOperativos]>
[sop@sistemas-~/SistOperativos]>
[sop@sistemas-~/SistOperativos]>chmod u+x VerArgumentos
[sop@sistemas-~/SistOperativos]>
[sop@sistemas-~/SistOperativos]>
[sop@sistemas-~/SistOperativos]>./VerArgumentos Sistemas Operativos Modernos
Nombre del script: ./VerArgumentos
Cantidad de argumentos: 3
Todos los argumentos: Sistemas Operativos Modernos
primer argumento: Sistemas
segundo argumento: Operativos
tercer argumento:Modernos
primer agumento despues de moverlos: Operativos
[sop@sistemas-~/SistOperativos]>
```

Figura 5: Ejemplo de argumentos posicionales y uso de las variables en el script

La variable $?

Esta variable es de sólo lectura y tiene como objetivo identificar estados de salida de los comandos ejecutados o características asociados a ellos a través de un código de retorno. Cuando un comando finaliza su ejecución devuelve al shell un número que indica si ha finalizado con éxito o no su ejecución. Si el estado de salida es cero 0, el comando se ejecutó con éxito, sino devolverá otro valor que depende de la causa que provocó la terminación anormal del comando. Podemos ver este valor con el comando:

```
echo $?
```

Ejemplo

```
[sop@sistemas-~/SistOperativos]>cat mostrar
        #Ejemplo de uso de los parámetros posicionales
                echo $3
                echo $1 $2
                more $4
[sop@sistemas-~/SistOperativos]>
[sop@sistemas-~/SistOperativos]>echo $?
0
[sop@sistemas-~/SistOperativos]>
[sop@sistemas-~/SistOperativos]>cat mostrados #Este archivo no existe
cat: mostrados: No existe el fichero o el directorio
[sop@sistemas-~/SistOperativos]>echo $?
1
[sop@sistemas-~/SistOperativos]>
```

Figura 6: Ejemplo de código de retorno que se guarda en la variable $?

Si necesitamos que un script retorne un código particular (siempre debe ser numérico), debemos incorporar al scripts la línea exit y el código de retorno que se desea. Ejemplo:

```
[sop@sistemas-~/SistOperativos]>cat mostrar
        #Ejemplo de uso de los parámetros posicionales
                echo $3
                echo $1 $2
                more $4
                exit 100
[sop@sistemas-~/SistOperativos]>
[sop@sistemas-~/SistOperativos]>./mostrar Sistemas Operativos UTNFRC /etc/networks
UTNFRC
Sistemas Operativos
default         0.0.0.0
loopback        127.0.0.0
link-local      169.254.0.0

[sop@sistemas-~/SistOperativos]>echo $?
100
[sop@sistemas-~/SistOperativos]>
```

Figura 5: Ejemplo de Retorno de un código particular, en este caso el valor 100.

Esta variable guarda el estado de salida del último comando que fue ejecutado. Esta variable no podrá recoger el estado de salida de un comando que se ejecutó en background (segundo plano).

Parámetros (argumentos) posicionales

Cuando el shell interpreta un comando, lo hace agregando nombres de variables a cada elemento de la línea de comandos. Estos elementos están separados por blancos (espacios). Las variables añadidas a los elementos en la línea de comandos son $0, $1, $2, etc. El nombre del comando es $0, el primer argumento o parámetro del comando es $1, y así con el resto.

Se puede acceder a tantos parámetros posicionales como el usuario desee, pero si se componen de más de un dígito éstos deberán estar encerrados entre llaves. Por ejemplo, ${15} representaría al parámetro posicional número quince. Las llaves son necesarias pues es posible concatenar el nombre de la variable con cualquier carácter, como vamos a ver más adelante. Veamos un ejemplo en la figura 6, para ello podemos crear el siguiente archivo con el comando *cat* o algún editor:

```
[sop@sistemas-~/SistOperativos]>cat mostrar
        #Ejemplo de uso de los parámetros posicionales
                echo $3
                echo $1 $2
                more $4
[sop@sistemas-~/SistOperativos]>
[sop@sistemas-~/SistOperativos]>
[sop@sistemas-~/SistOperativos]>chmod 744 mostrar
[sop@sistemas-~/SistOperativos]>
[sop@sistemas-~/SistOperativos]>./mostrar Buenos días estudiantes /etc/networks
estudiantes
Buenos días
default         0.0.0.0
loopback        127.0.0.0
link-local      169.254.0.0
```

Figura 6: Ejemplo de uso de los parámetros posicionales.

Si deseamos indicarle al shell que un conjunto de caracteres separados por blancos representa un elemento, deberemos encerrarlos entre comillas.

Variables de entorno del usuario

Las variables de entorno guardan información importante y necesaria para los programas que se ejecutarán y para el shell mismo. Incluso parte del proceso de conexión al sistema, por parte del usuario, consiste en la creación de su entorno. Por ejemplo: algunos programas necesitarán que la ruta de acceso al archivo de ayuda esté guardada en alguna variable; o cuando ejecutemos un comando, el shell necesitará saber la ruta de acceso al directorio donde están los archivos ejecutables; o algunas otras variables especifican configuraciones personalizadas, etc. Usted puede ver el valor predeterminado de las variables de entorno simplemente utilizando el comando *set o env,* cuyo uso se detalla más adelante.

Variable	Descripción
HOME=/home/login	Configura su directorio de usuario, es decir, la localización desde donde inicia la sesión. Sustituya *login* por su identificador de entrada al sistema, por ejemplo */home/mgarcia.*
PATH=path	Contiene una lista de directorios que el shell examina al buscar archivos ejecutables.
PS1=prompt	Contiene el identificador primario de shell, por defecto es el signo $.
SHELL=shell	Indica la ruta absoluta al programa intérprete de comandos. Por ejemplo, /bin/bash.
TERM=termtype	Especifica el terminal por defecto. Suele ser xterm.
PWD=/root	Especifica la ruta por defecto del usuario.

Tabla. Algunas variables de entorno.

PATH	la variable PATH contiene una serie de campos separados por dos puntos (:), en donde cada campo es un directorio que contiene comandos o archivos ejecutables.

Para ver el contenido de la variable PATH del usuario, ejecutamos el comando:

```
sop@Sistemas:/$echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
sop@Sistemas:/$
```

Figura 7: contenido de la variable de entorno PATH

El orden en el que están definidos los directorios en la variable, es importante, pues es en ese orden en el que el shell buscará un archivo ejecutable, y sólo lo hará en los directorios definidos en PATH. Es decir que al ejecutar un comando, el shell buscará el archivo primero en /usr/local/bin, luego en /usr/sbin, /bin, /usr/local/games y por último en /usr/games. Si el directorio en el que se encuentra el comando no está definido en PATH, el sistema nos informará que no lo encuentra.

Si pretende crear sus propios comandos, debe incluir en la variable PATH, la ruta de acceso al directorio que contiene dichos comandos. Generalmente /usr/local/bin contiene comandos locales creados por el usuario. Puede utilizar la siguiente línea para agregar su directorio de comandos a la variable:

PATH=/home/sop/ejecutables:$PATH

Con una simple asignación colocamos como primer directorio de búsqueda a */home/sop/ejecutables*, o podríamos colocarlo como último directorio de búsqueda con:

PATH=$PATH: /home/sop/ejecutables

Debemos tener en cuenta, no crear scripts con nombres de comandos que ya existen, como *date*, *cp* o *find* pues si el directorio de nuestros ejecutables es el último en la lista definida en la variable PATH, nuestros archivos nunca van a ejecutarse ya que el shell examina primero el directorio */bin,* antes que nuestro directorio.

Uso de comandos adicionales

env	ejecuta un script con un entorno modificado pues es posible pasarle como argumento el nombre y nuevo valor de una o varias variables del Shell.

Este comando es un archivo externo al shell, su trayectoria es */usr/bin/env.*

Sintaxis:

```
env [-i] [--unset variable] [variable=valor] [script [parámetros ...]]
```

Opciones:

[--unset variable]	elimina la variable del entorno, si es que ésta existe.
[variable=valor]	cambia el contenido de la variable de entorno por el nuevo valor, el cual puede estar vacío, por ejemplo: 'variable='

[script [parámetros ...]] especifica el script a ejecutar, éste se busca de acuerdo a lo determinado por la variable de entorno PATH. Cualquier parámetro posterior pasa al script como parámetro propio.

[-i] Comienza con un entorno vacío, ignorando el heredado.

El comando *env* sin opciones muestra una lista de las variable de entorno y sus valores actuales.

Ejemplo

```
$ env
```

nos mostrará un listado parecido al siguiente:

```
LESSOPEN=| /usr/bin/lesspipe.sh  %s
ENV=/root/.bashrc
COLORTERM=gnome-terminal
HISTSIZE=1000
HOSTNAME=localhost.localdomain
LOGNAME=mgarcia
MAIL=/var/spool/mail/mgarcia
HOSTTYPE=i386
SHELL=/bin/bash
HOME=/rhome/mgarcia
      :
```

```
[sop@sistemas-~/SistOperativos]cat ejemplo
echo $variable
echo $SHELL
[sop@sistemas-~/SistOperativos]./ejemplo

/bin/bash
[sop@sistemas-~/SistOperativos]env SHELL=/bin/csh variable=valor ./ejemplo
valor
/bin/csh
[sop@sistemas-~/SistOperativos]./ejemplo

/bin/bash
[sop@sistemas-~/SistOperativos]
```

Figura 8: Ejemplo de uso de ejecución de un programa pero con otras variables de entorno.

Si prestamos atención en el ejemplo de la Figura 8, el contenido de variable es vacío y el de SHELL es /bin/bash, en el entorno en el que lo estamos ejecutando, pero al ejecutar la línea:

```
$ env SHELL=/bin/csh variable=valor ./ejemplo
```

Podemos apreciar que el programa ejemplo se ejecuta en un entorno distinto ya que el contenido de la variable es valor y el de SHELL es /bin/csh, lo que significa que este programa se ejecutó PS1 en el SHELL csh y no en bash.

printenv imprime todos o una lista de variables de entorno pasadas como argumento.

Sintaxis:

```
printenv  variables ...]
```

El estado de salida del comando es:

0 si se encontraron todas las variables especificadas.
1 si no se encontró al menos una de las variables especificadas.
2 si ocurrió un error de escritura.

Ejemplo

```
$ printenv  HOME  SHELL

/rhome/mgarcia
/bin/bash
```

El comando muestra el contenido de las variables HOME y SHELL. Por supuesto, el comando *echo $?* mostrará el valor 0.

set el comando *set* muestra tanto las variables locales como las variables de entorno y sus valores correspondientes.

Ejemplo

```
sop@Sistemas:/$set |more
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extqu
ote:force_fignore:interactive_comments:login_shell:progcomp:promptvars:source
path
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_COMPAT_DIR=/etc/bash_completion.d
BASH_LINENO=()
BASH_REMATCH=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="3" [2]="30" [3]="1" [4]="release" [5]="i586-pc-li
nux-gnu")
BASH_VERSION='4.3.30(1)-release'
COLORTERM=xfce4-terminal
COLUMNS=77
COMP_WORDBREAKS=$' \t\n"\'><=;|&(:'
DIRSTACK=()
DISPLAY=:0.0
EUID=0
GROUPS=()
HISTFILE=/root/.bash_history
HISTFILESIZE=500
HISTSIZE=500
HOME=/root
HOSTNAME=simred
HOSTTYPE=i586
IFS=$' \t\n'
LANG=es_AR.UTF-8
LANGUAGE=es_AR:es
LINES=31
```

unset	asigna a una variable creada por el usuario el valor NULL

Este comando también permite inhabilitar una función, es de gran utilidad si estamos trabajando con un script demasiado largo, eliminamos una variable simplemente utilizando el comando *unset*.

Sintaxis:

```
unset  [-f]  [lista de variables/ lista de funciones]
```

La opción -f indica que los argumentos hacen referencia a funciones y no a variables. Las variables de entorno como PS1, HOME, etc, no pueden ser inhabilitadas.

export	exporta el valor de una variable para que pueda ser accesible en un subshell

Al crear una variable, ésta queda definida sólo para el shell que se está ejecutando. Las variables definidas en la línea de comando podrán ser utilizadas sólo por los comandos tipeados en ella. Es decir que una variable definida en un script, al concluir éste, pierde su valor. El comando export, permite que una variable definida en un shell, pueda ser utilizado por los shells posteriores.

Sintaxis:

```
export [lista de variables...]
```

Ejemplo

Si creamos el siguiente script:

```
$mes=Agosto
$echo $mes
```

Al ejecutarlo nos mostrará la cadena *Agosto.* Luego de la ejecución si ingresamos en la línea de comando *echo $mes*, veremos que la variable está vacía. Si agregamos al script la siguiente línea:

Ejemplo

```
$export mes
```

Luego de ejecutar el script, el comando *echo $mes* nos mostrará la cadena Agosto. El comando *export,* sin argumentos, muestra las variables de entorno que están configuradas en su shell.

Ejemplo

```
#echo $variable

#variable=valor
#echo $variable
valor
#
#cat programa
echo $variable
#./programa

#export variable
#./programa
valor
#
```

Figura 9: Ejemplo de comando export

En este ejemplo se puede apreciar como al crear una variable en el shell, la misma no está disponible en el subshell al ejecutar el script programa, la linea siguiente a la ejecución esta vacia. Luego de ejecutar el comando export la variable está disponible en el subshell y en la linea siguiente a la ejecución se muestra el contenido de la variable.

Líneas de órdenes

Las líneas de órdenes son una agrupación que se pueden ejecutar en forma: secuencial, condicional o con encadenamientos de tuberías.

Secuencia de órdenes

Es un conjunto de órdenes que se pasan al shell en una línea, separadas por punto y coma. El shell las considera como órdenes individuales y las ejecuta secuencialmente.

Sintaxis:

```
orden1; orden2; orden3
```

Ejemplo

Suponga que desea realizar una copia del archivo *morosos* (que existe), luego ver el número de i-nodo del nuevo archivo y posicionarse en el directorio /etc, por lo que deberá plantear la siguiente secuencia:

```
[sop@sistemas-~/SistOperativos]cp morosos morososcp; ls -i morososcp; cd /etc
268822 morososcp
[sop@sistemas-/etc]
```

Figura 10: Ejemplo de secuencia de comandos.

Luego de copiar el archivo *morosos* con el nombre *morososcp*, el usuario queda ubicado en el directorio /etc.

Órdenes Condicionales

Las órdenes condicionales son dos órdenes relacionadas por conectivos lógicos, AND y OR, dónde la ejecución de la segunda orden depende de la ejecución de la primera.

Después de su ejecución toda orden de Linux genera un código de retorno, que indica si la orden se ha ejecutado con éxito o ha sucedido algún tipo de error, como puede ser el no encontrar un archivo. Linux guarda el código de retorno en la variable **?**, que será igual a cero o distinto de cero según se haya ejecutado con éxito o no el comando.

Sintaxis:

```
orden1 && orden2
```

En ésta línea de comandos se utilizó el conectivo AND (&&) por lo que la *orden2* se ejecuta sólo cuando la *orden1* se ha ejecutado con éxito.

```
orden1  ||  orden2
```

En ésta línea de comandos se utilizó el conectivo OR (||) por lo que la *orden2* se ejecuta sólo cuando la *orden1* no se ha ejecutado con éxito.

Ejemplo

```
[sop@sistemas-~/SistOperativos]cp  listado  nomina &&  more  nomina
"Este es el contenido de Listado"
[sop@sistemas-~/SistOperativos]
```

Figura 11: Ejemplo de orden condicional &&.

La primera orden copia el archivo *listado* a *nomina*, si la copia se realiza con éxito se ejecuta la segunda orden, es decir, que la salida que genera esta línea de comandos es el contenido del archivo *nomina,* si es que pudo copiarse *listado* a *nomina*.

Supongamos ahora que no estamos seguros si el archivo *listado* está en el directorio actual o en el directorio padre. En ese caso para copiar el archivo podríamos utilizar la siguiente orden:

```
[sop@sistemas-~/SistOperativos]cp  listado  nominaLocal || cp   ../listado  nominaPadre
[sop@sistemas-~/SistOperativos]ls nomi*
nominaLocal
[sop@sistemas-~/SistOperativos]
```

Figura 12: Ejemplo de orden condicional ||.

Si la primera orden no tiene éxito se ejecuta la segunda orden, que busca al archivo *listado* en el directorio padre y lo copia como *nominaPadre* en el directorio actual.

Grupo de órdenes

Es una secuencia de órdenes encerradas entre paréntesis, que el sistema ejecuta secuencialmente pero en cuanto a las entradas y las salidas, son tratadas como una sola orden. La variable de retorno tendrá el código de retorno de la última orden de la secuencia.

Sintaxis:

```
(orden1 ; orden2 ; orden3)
```

Sigamos con el ejemplo del punto anterior, copiar el archivo *listado* como *nomina* y luego mostrarlo por pantalla. Sin embargo, no estamos seguros de la ubicación del archivo *listado*, por lo que planteamos la *orden* copiar, como un grupo de órdenes que si tiene éxito, permite la ejecución de la *orden* mostrar.

```
[sop@sistemas-~/SistOperativos](cp  listado  nomina || cp   ../listado  nomina) && more nomina
"Este es el contenido de Listado"
[sop@sistemas-~/SistOperativos]
```

Figura 13: Ejemplo de grupo de ordenes.

Como grupo de órdenes, si la primera orden no tiene éxito se ejecuta la segunda orden, que busca al archivo *listado* en el directorio padre y lo copia como *nomina* en el directorio actual. Luego, si la copia se realiza con éxito se ejecuta la tercer orden, es decir, que la salida que genera esta línea de comandos es el contenido del archivo *nomina,* si es que pudo copiarse *listado* a *nomina.*

Para comprender mejor su funcionamiento planteamos el siguiente ejemplo:

Ejemplo

```
$ (ls  /dev;pwd) > dispositivos
```

Las salidas de los comandos *ls* y *pwd*, que se ejecutan en forma secuencial, se redireccionan al archivo *dispositivos*. Es decir que este archivo tendrá el contenido del directorio dev y path al directorio actual.

Si no estuvieran los paréntesis, si se hubiera planteado la línea simplemente como una secuencia de comandos:

Ejemplo

```
$ ls /dev; pwd  > dispositivos
```

Sólo la salida de la segunda orden se redirecciona al archivo *dispositivos.*

Tubería (pipeline)

En este tipo de líneas de órdenes la salida de una orden es entrada de la siguiente, en donde:

Sintaxis:

```
$ orden1 | orden2 | orden3
```

El pipe es un buffer donde la orden deposita su salida, y desde donde el siguiente comando toma su entrada.

Ejemplo

```
$ who | grep pepe
```

El comando who genera un listado de los usuarios conectados al sistema, que deposita en el pipe, grep busca en ese listado la cadena de caracteres *pepe* y muestra por pantalla las líneas en donde la encuentra. Con esta orden podemos ver si el usuario pepe está conectado al sistema, en qué terminales está trabajando y a qué hora se conectó.

Estructuras de Control

Existen dos tipos de estructuras de control básicas:

- De decisión
- Repetitivas o Iterativas

Podemos identificar como estructuras de decisión a **if – fi** y **case – esac,** y como estructuras repetitivas a **while** y **for.**

La estructura condicional if – fi

Es una estructura de condición alternativa pues permite seleccionar entre dos posibilidades de acción, si la *condición* ha tenido éxito. Esto es, si el valor de status o retorno del comando ejecutado como condición, ha sido cero. Si la condición está compuesta por un conjunto de comandos, la estructura if considera el status del último comando.

Sintaxis:

```
if condición
then
      comandos
else
      comandos
fi
```

La proposición **else,** es decir, la ejecución de la rama falsa de la estructura, es opcional.

Recuerde que cada comando devuelve un valor de retorno al shell, para indicar en qué condiciones se ha ejecutado. Si el retorno es un "0" (verdadero), se entiende que el comando se ejecutó en forma exitosa, y si retorna otro valor (falso) se interpreta como que hubo un error, y precisamente el número retornado, indica en qué situación se produjo tal error. La proposición **if** interpreta el "0", como un verdadero, y un número distinto de "0", será interpretado como falso. La condición de esta estructura siempre debe ser un comando o un conjunto de comandos.

Ejemplo

```
#!/bin/bash
# Objetivo: enviar un mensaje al usuario monica, si está conectado
if who |  grep  monica
then
      write  monica
else
      echo "el usuario monica aún no ha llegado"
fi
```

El comando who, que muestra un listado de los usuarios conectados al sistema, envía su salida a la tubería dónde el comando busca la cadena “monica”, que es el nombre del usuario con el cual debemos comunicarnos. Si el comando tiene éxito, se ejecuta la rama verdadera del condicional (then), en caso contrario se muestra un mensaje por pantalla.

La estructura condicional case

Es una estructura de condición que permite seleccionar una entre varias posibilidades de acción, según sea el valor de la variable. Compara el valor de una única variable con los patrones especificados en su estructura. Si encuentra algún patrón coincidente al valor de la variable, ejecuta la lista de comandos asociada a ese patrón, y luego la secuencia del programa continúa fuera de la estructura.

Sintaxis:

```
case $variable in
      patrón1)comando1
              comando2;;
      patrón2)comando3
              comando4;;
            :
            :
      *) acción por defecto ;;
esac
```

Case compara la variable con los patrones, desde arriba hacia abajo y ejecuta los comandos asociados, con el primer patrón que reconozca y luego sale de la estructura. Los patrones son cadenas de caracteres, en donde es posible incorporar algunos metacaracteres, definidos para la proposición **case**, específicamente. Uno de los usos del **case**, es el de validar o reconocer argumentos de un comando.

Ejemplo

```
#!/bin/bash
# Objetivo: mostrar un menú de actividades al usuario
echo " Por favor seleccione una opción para:"
echo " R Ver pahtname"
echo " W Muestra usuarios conectados"
echo " S Sale del programa"
#el usuario debe ingresar una opción que será depositada en var
read var
case $var  in
      R|r) echo "El pathname: "pwd;;
      W|w) echo "Los usuarios conectados: "
            who;;
      S|s) echo "opción Salir";;
esac
```

Al definir los patrones se puede utilizar el metacaracter | para la función lógica " O". También podemos utilizar los metacaracteres ***** y **[...]**. El "*" como patrón, se usa para indicar la acción a desarrollar, en el caso de que la variable analizada por el case, no sea ninguna de las previstas por los patrones anteriores.

La estructura repetitiva for

Esta estructura ejecuta el cuerpo de acciones un número finito de veces. Este comando puede ejecutarse tanto en un programa del shell, como desde la línea de comandos.

Sintaxis:

```
for  variable in lista_de argumentos
do
      comandos1
      comandos2
      :
done
```

Se puede usar cualquier variable (generalmente se utiliza la variable "i"), la cual adoptará en cada iteración un elemento de la lista de argumentos.

Ejemplo

```
#!/bin/bash
# el objetivo del script es ver el contenido
# de todos los archivos del directorio actual

for i in `ls`
do
      more  $i
done
```

En este script, la ejecución del comando *ls* produce un listado de los archivos existentes en el directorio actual. La variable **i**, en la primera iteración adoptará como valor el nombre del primer archivo de la lista que genere *ls*, en la segunda iteración adoptará el segundo nombre de la lista, y así sucesivamente. Por lo que el número de iteraciones dependerá de la cantidad de archivos que hayan sido listados. La acción que se repite, es mostrar el archivo cuyo nombre está en la variable **i**. Una forma simple de escribir el comando, es la siguiente:

Sintaxis:

```
for i  in lista_de argumentos; do comandos; done
```

Ejemplo

```
for i in `ls`; do more  $i ; done
```

La estructura repetitiva while

Esta instrucción se utiliza para repetir una serie de comandos varias veces, de acuerdo a una condición que controla el ciclo. Es decir mientras la condición, *sea verdadera*, el cuerpo de acciones se ejecuta.

Sintaxis:

```
while  condición
do
      comando1
      comando2
         :
done
```

En cada iteración, **while** ejecuta la condición, que es un comando. Si este comando concluye con éxito (retorna un "0", o sea un verdadero), se ejecutan los comandos del cuerpo del while. Si el retorno del comando es un falso, finaliza el ciclo, y se continúa ejecutando la instrucción siguiente a **done**. La sentencia **done** indica el final del cuerpo de acciones, y el programa regresa al comienzo del **while**, y se ejecuta de nuevo el comando de condición. Analicemos el siguiente ejemplo:

Ejemplo

```
#!/bin/bash
#Realizar un Menú que al ingresar un carácter informe si es una letra,
#un valor numérico del 1 al 9 otros sino es ninguno de los anteriores
#y con 0 salir.
clear
while true
do
      echo "Ingrese un caracter:"
      read valor
      case $valor in
            [a-z,A-Z]) echo "Ud. ingreso una letra.";;
            [1-9]) echo "Ud. ingreso un número del 1 al 9.";;
            0) break;;
            *) echo "Ud. ingreso un carácter no valido.";;
      esac
done
```

La condición del ciclo while es el comando true, lo cual genera un bucle sin fin sino se coloca un condicional que termine el bucle. El ciclo se repite hasta que se ingresa el valor 0.

Comandos y órdenes

test	permite comprobar el valor de cualquier expresión

Este comando permite comprobar el valor de cualquier expresión. *test* devuelve un 0, si la expresión se evalúa como verdadera (true) y, devuelve un 1 si la expresión es falsa.

Sintaxis:

```
test expresión
```

Las expresiones pasadas a test como argumentos, pueden combinarse mediante los siguientes operadores lógicos:

|| (OR) devuelve true si cualquiera de las expresiones es verdadera.
&& (AND) devuelve true si ambas expresiones son verdaderas.
! (NOT) devuelve el valor opuesto de la expresión.

Algunos argumentos habituales son:

!expresión Entrega el valor opuesto de la expresión. Si la expresión es *true*, se devolverá *false.*
(expresión) && (expresión2) Entrega un valor verdadero si ambas expresiones son verdaderas.
(expresión) || (expresión2) Entrega un valor verdadero si cualquiera de las expresiones es verdadera.

El comando test puede combinarse con un conjunto de operadores del shell. Existen varios tipos de operadores:

- De verificación de archivos
- De comprobación de cadenas
- Aritméticos

Operadores de verificación de archivos

Sintaxis:

```
test opción nombre_archivo
```

Las siguientes opciones devuelven el valor verdadero (true) si:

-a Existe el archivo.
-b Existe el archivo y es un archivo especial de bloque.
-c Existe el archivo y es un archivo especial de carácter.
-d Existe el archivo y este es un directorio.
-s El archivo existe y su tamaño es mayor que cero.
-f El archivo existe y es un archivo regular.
-r El archivo existe y el usuario tiene permiso de lectura.
-w El archivo existe y el usuario tiene permiso de escritura.
-x El archivo existe y el usuario tiene permiso de ejecución.

Operadores de comprobación de cadenas

Sintaxis:

```
test [opción cadena] [cadena operador cadena]
```

Las siguientes opciones devuelven el valor verdadero (true) si:

-z La cadena es nula
-n Existe la cadena

Operadores	
cadena1=*cadena2*	cadena1 es igual a la cadena2
cadena1==*cadena2*	Las dos cadenas son iguales.
cadena1 != *cadena2*	cadena1 es distinta a la cadena2
cadena	La cadena no es nula
cadena1<*cadena2*	cadena1 tiene una ordenación anterior que cadena2.
cadena1>*cadena2*	cadena2 tiene una ordenación anterior que cadena1.

Operadores Aritméticos

Sintaxis:

```
test entero1 operador entero2
```

Los siguientes operadores devuelven el valor verdadero (true) si:

Operadores	
numero1 ***-eq*** *numero2*	ambos enteros son iguales.
numero1 ***-ne*** *numero2*	los enteros son distintos.
numero1 **-gt** numero2	numero1 es mayor a numero2
numero1 ***-ge*** *numero2*	numero1 es mayor o igual a numero2
numero1 ***-lt*** *numero2*	numero1 es menor a numero2
numero1 ***-le*** *numero2*	numero1 es menor o igual a numero2

El valor que devuelve test se utiliza en combinación con las estructuras de control ya vistas.

Ejemplo

```
If test –d prueba
then
      rm –r  prueba
fi
```

El script anterior permite borrar un directorio en forma recursiva si es que este existe.

Ejemplo

```
If test  $SHELL=/bash/csh
then
      programa    #ejecuta el comando programa
else
      env  SHELL=/bin/csh  programa
fi
```

La proposición ***if*****,** ejecuta el comando *test*, y analiza su retorno. A su vez, el comando *test* analiza la *condición* que utiliza un operador de comparación de cadenas, y si ésta es verdadera retorna un "0", y si es falsa retorna un "1". En el ejemplo anterior se desea ejecutar el archivo programa con el shell csh y, si éste no está definido por defecto, lo modifica con el comando *env.*

break	Se utiliza para salir de un bucle y se ejecuta la orden que sigue inmediatamente a éste.

Sintaxis:

```
break [n]
```

La opción **n** indica el número de niveles o bucles de los que hay que salir.

Ejemplo

```
for  i in  lunes martes miercoles jueves
do
      if  test  $i = miercoles
      then
            break
      else
            echo "Hoy es "$i
fi
done
```

La ejecución del script mostrará la siguiente salida:

```
lunes
martes
```

cuando la variable i contenga la cadena miércoles, la orden break sale del bucle for.

continue	este comando salta a la siguiente iteración, sin salir del bucle

Permite evitar la ejecución del cuerpo de acciones para ciertos valores de la variable que controla el bucle.

Sintaxis:

```
continue [n]
```

La opción **n** indica el bucle al cual deberemos saltar. Si se le da un argumento numérico, el control va hacia el bucle indicado por dicho número.

Ejemplo

```
$continue 5
```

Va hacia el comienzo del quinto bucle que engloba dicha orden.

let	permite operar con variables numéricas de un modo sencillo en los scripts

Ejemplo

```
!#/bin/bash
var1=5
let var1=var1+2
```

El resultado de este script será la suma del contenido de la variable var1+2 en este caso 7. También se puede realizar la operación var1+=2, daría el mismo resultado.

Comando para operaciones aritméticas

expr	se utiliza para resolver operaciones aritméticas sencillas.

Solo se debe pasar como argumentos los dos valores de la operación que se desea realizar y el operador.

Sintaxis:

```
expr [operandoA] operador [operandoB]
```

Los operadores aritméticos

- **+** Suma
- **-** Resta
- ***** Multiplicación (la barra invertida es necesaria porque el * es un carácter especial)
- **%** Modulo
- **/** división

Ejemplo

```
sop@Sistemas:/$expr 8 + 2
10
sop@Sistemas:/$Numero1=7
sop@Sistemas:/$Numero2=8
sop@Sistemas:/$
sop@Sistemas:/$expr $Numero1 + $Numero2
15
sop@Sistemas:/$
sop@Sistemas:/$expr $Numero1 \* $Numero2
56
sop@Sistemas:/$
sop@Sistemas:/$multiplicacion=$(expr $Numero1 \* $Numero2)
sop@Sistemas:/$echo $multiplicacion
56
sop@Sistemas:/$
sop@Sistemas:/$expr $multiplicacion / $Numero1
8
sop@Sistemas:/$
sop@Sistemas:/$expr $multiplicacion % $Numero2
0
sop@Sistemas:/$
```

En las operaciones realizadas, se puede apreciar una suma de valores constantes, el uso de variables y los distintos operadores.

eval obtiene el valor de los argumentos de la línea de comandos y ejecuta los mismos como comandos.

Ejemplo

```
[sop@sistemas-~/SistOperativos]cat ejemplo2.sh
#!/bin/sh

#Uso de comillas dobles-->las variables son sustituidas
Comando="ls -l $data"
echo $Comando

#Uso de comillas simples-->las variables no son sustituidas
#y son tratadas como cadenas de caracteres
Comando='ls -l $data'
echo $Comando

data="/media"
eval echo $Comando
[sop@sistemas-~/SistOperativos]
[sop@sistemas-~/SistOperativos]
[sop@sistemas-~/SistOperativos]./ejemplo2.sh
ls -l
ls -l $data
ls -l /media
[sop@sistemas-~/SistOperativos]
```

Figura 14: Ejemplo de sustitución de variables

```
[sop@sistemas-~/SistOperativos]cat ejemplo2.sh
#!/bin/sh

#Uso de comillas dobles-->las variables son sustituidas
data="/opt"
Comando="ls -l $data"
echo $Comando
$Comando

#Uso de comillas simples-->las variables no son sustituidas
#y son tratadas como cadenas de caracteres
Comando='ls -l $data'
echo $Comando
$Comando

data="/media"
eval echo $Comando
eval $Comando
[sop@sistemas-~/SistOperativos]./ejemplo2.sh
ls -l /opt
total 0
ls -l $data
ls: no se puede acceder a $data: No existe el fichero o el directorio
ls -l /media
total 8
lrwxrwxrwx 1 root root      6 nov 18  2014 cdrom -> cdrom0
drwxr-xr-x 2 root root   4096 nov 18  2014 cdrom0
drwxrwsr-x 2 root vboxsf 4096 abr  1  2015 vbox
[sop@sistemas-~/SistOperativos]
```

Figura 15: Ejemplo de ejecución utilizando el comando eval

En este caso eval evalúa el contenido del argumento y lo ejecuta como comando.

Declaración de funciones

Todas las funciones deben ser declaradas antes de que puedan ser utilizadas.

Sintaxis:

```
function nombrefuncion(){
  Commands
}
```

Para invocar a la función solo es necesario el nombre de la función.

Sintaxis:

```
nombrefuncion [argumentos…]
```

Le podemos pasar argumentos. Los argumentos que se pasan a las funciones, se comportan de la misma manera que los argumentos suministrados a un script.

Ejemplo

```
#!/bin/bash
 function Imprimir {
    echo $1
    }
 Imprimir Hola
 Imprimir Mundo
```

Actividad 1

1. Comente cuál es el contenido de cada una de las siguientes variables. Explique las diferencias.

```
cont1=pwd
cont2=”pwd”
cont3=‘pwd‘
cont4=´pwd´
cont5=’pwd’
```

2. Cree un script que solicite sus datos personales y los muestre.

```
# en este script utilizaremos el comando read
clear          #limpia la pantalla
echo  -n   “Ingrese su nombre completo:  “
read   nbre
echo  -n   “Ingrese su Apellido:  “
read  ap
echo –n “Ingrese su edad:  “
read ed
echo
echo “Los datos completos son:”$nbre   $ap    $ed
```

Recuerde que para ejecutar puede tipear alguna de las siguientes órdenes:

- sh nom_scrpts
- chmod +x nom_scripts
- ./nom_scripts

3. Modifique el indicador del shell en la variable PS1 para que muestre:

a. su nombre de usuario.

```
PS1=”$PWD\$”
```

b. su login-name

```
PS1=”$LOGNAME\$”
```

c. una cadena.

```
PS1=”Listo\$”
```

4. Cree un script que le permita copiar un archivo (utilice parámetros posicionales).

a. creamos el archivo con la orden *cat > copiar*

```
cp  $1 $2
```

Lo ejecutamos como: sh copiar archivoa archivob

De esta manera al ejecutar el programa copiar, le pasamos dos argumentos o parámetros posicionales, $1 y $2, que corresponden a los nombres de los archivos con los cuales vamos a trabajar. Si los archivos no están en el directorio actual recuerde incluir en el nombre del archivo, la ruta de acceso al mismo.

5. Cree el script saludo que le de la bienvenida cuando usted comience una nueva sesión, y además le muestre el shell que utiliza, la fecha y hora de conexión.

```
# este script saluda al usuario en cada nueva conexión
echo  “Saludos amigo  “$LOGNAME
echo  “Espero se cumplan sus expectativas”
echo
echo “el shell utilizado es : “$SHELL
echo “Hoy es:  “ ‘date‘
echo “Hora:  “ ‘time‘
```

6. Para que el archivo anterior se ejecute en cada nueva sesión, agréguelo al archivo .bash_profile y luego ejecútelo.

```
vi .bash_profile
```

7. Muestre un listado de las variables de entorno del usuario con sus respectivos valores. env

8. Por la línea de comandos cree la variable ed=20 y luego muestre su contenido.

```
ed=20
echo $ed
```

9. Defina a la variable ed como de sólo lectura.

```
declare –r ed     #define a la variable como de sólo lectura.
echo $?      #verificamos si el comando anterior tuvo éxito.
ed=40        #asignamos un nuevo valor a la variable
```

Verá Ud. que el shell le muestra un mensaje de error, por lo que la variable ed conserva el valor 20.

10. Ejecute el script creado en el punto 5, pero con el shell csh y nombre de usuario mtoledo.

```
env  SHELL=/bin/csh  LOGNAME=mtoledo  ./saludo
```

Actividad 2

1. Cree el script control que emita un mensaje si usted tiene más de tres terminales abiertas.

```
m=‘(who | grep $LOGNAME) | wc –l ‘
if  test  $m  -gt  3
 then
       echo  “Ud. Tiene demasiadas terminales abiertas”
else
       echo “ la cantidad de terminales abiertas es correcta”
fi
```

2. Modifique el script anterior de forma que pueda controlar las terminales abiertas de cualquier usuario.

```
echo  -n  “Ingrese el nombre del usuario:  “
read  us
m=‘(who | grep $us) | wc –l ‘
if  test  $m  -gt  3
 then
       echo  “$usted tiene demasiadas terminales abiertas”
else
       echo “ la cantidad de terminales abiertas es correcta”
fi
```

3. Cree el script enlace al que deberá pasarle dos parámetros posicionales (el archivo a enlazar y el nombre del enlace) si no es así, el script tendrá que solicitarle esos datos.

```
if  test  $#  -eq  2
 then
 #el usuario pasa al script dos parámetros
       echo $@          #muestra los parámetros posicionales
       ln  $1  $2
else
       echo –n "Ingrese  el nombre del archivo a enlazar: "
       read  a
       echo  -n  "Ingrese el nombre del enlace: "
       read  b
       # verifico si el archivo a enlazar existe
       if  test  -a $a
       then
             ln  $a  $b
       else
             echo  $a  "no existe"
       fi
fi
```

4. Cree el script opciones que le permita seleccionar entre las siguientes actividades:
1. ver información sobre el espacio en disco que ocupan los sistemas de archivos montados
2. ver un informe sobre la memoria libre y usada
3. salir del script.

```
While true
do
       clear
       echo
       echo "Elija una opción del siguiente menú: "
       echo  "D    utilización del disco
       echo  "F    utilización de la memoria"
       echo  "S    salir"
       read var
       case  $var  in
             D|d)  df
       read a;;
             F|f) free
       read a;;
             S|s)  exit
       esac
done
```

5. Cree el script contar que cuente y muestre las líneas de los archivos de un directorio específico.

```
echo  -n "Ingrese el nombre del directorio: "
read a
if  test  -d  $a
 then
       for i in  'ls  -1 $a'
       do
             if  test  -f  $a/$i
             then
                   con='wc  -l  $a/$i '
                   echo  $con
                   read
             fi
       done
else
       echo  $a  "no es un directorio"
fi
```

6. Realizar un programa que reciba por parámetro un texto y determine si es archivo o si es un directorio, en caso contrario informe que no es ninguno de los dos casos. Utilice una función para informar el mensaje que no cumple con ser archivo, ni directorio.

```
#!/bin/bash

Mensaje(){
       echo "El parámetro recibido no cumple con ser un archivo o
       directorio."
       }

if test $# -lt 1
 then
       Mensaje
       elif test -f $1
       then
             echo "$1 es un archivo."
       elif test -d $1
       then
             echo "$1 es un directorio."
       else
       echo "$1 no es archivo, ni directorio."
 fi
```

Agregue un comentario al lado de cada línea del if que indique que se está evaluando como condición.

7. Escribir un script que muestre la tabla de multiplicar hasta 5.

```
#!/bin/bash
clear
echo "Ud. Eligio ver la tabla del: $1"
multiplicador=0
while test $multiplicador -le 5
do
       resultado=`expr $multiplicador \* $1`
       echo $1*$multiplicador = [ $resultado ]
       multiplicador=`expr $multiplicador + 1`
done
```

8. Crear un programa que permita identificar el tamaño en MB del archivo que se pasa por parámetro. Utilice una función para informar si el valor que se recibió por parámetro no es un archivo.

```
#!/bin/bash
Mensaje(){
             echo "El valor ingresado no es un archivo."
            }
if test $# -lt 1
       then
             if test -f $1
                   du -h $1
             else
                   Mensaje
             Fi
fi
```

9. Realice un script que haga una copia de seguridad de su home cada 1 minuto. La copia que se realice deberá ser copiada a un directorio que tenga la hora y fecha del momento en que se lleva a cabo. Deberá usar cron para poder cumplir con esta actividad.

En el cron tendremos la siguiente entrada:
*/1 * * * * /home/<user>/<programa>

Programa

```
#!/bin/bash
cp -r /home/<user>/ /home/<user>/$(date  +%d%m%Y_%H%M%S)
```

Actividad 3

1. ¿Qué es la programación del shell? ¿Para qué la utilizamos?
2. ¿Cómo se asigna un valor determinado a una variable del shell?
3. Defina qué es un parámetro posicional.
4. Explique las dos formas en que se puede ejecutar un shell script.
5. Indique el nombre de al menos tres variables de entorno y su contenido.
6. Desarrollar un programa que genere un archivo con una lista de los sudirectorios que hay dentro un directorio cualquiera, usando parámetros posicionales.

Prácticos Integradores

LINUX: PRÁCTICO INTEGRADOR N° 1

Temas: File system. Rutas. Procesos. Filtros. Enlaces. Tubería. Redirección E/S. Permisos. Metacaracteres.

---

1. Cree, en su directorio de login los subdirectorios *trabajo* y *practico*. En el directorio *trabajo* crear los subdirectorios *unidad1* y *unidad2*.
2. Cree en el subdirectorio *unidad1* el archivo *terminales*, con información de los archivos de /*dev* cuyos nombres comiencen con los caracteres **pt**. Trabaje sin cambiarse de directorio.
3. Cree el archivo *números* cuya información sea el contenido del directorio /*etc*, en formato extendido, incluido el n° de i-nodo de los archivos.
4. Copie las líneas del archivo *numeros* que tengan la cadena de caracteres "**rwx-----x**" al archivo *permisos* en el directorio *practico.*
5. En reemplazo de los puntos 3 y 4, cuál sería la línea de comandos necesaria para crear el archivo *permisos* en el directorio *practico* con una sola línea de comando, es decir sin crear el archivo intermedio *numeros***.**
6. Posiciónese en el directorio *unidad2*. Muestre por pantalla en forma paginada el directorio /*home*. Utilice nombre relativo y pipeline.
7. Busque el archivo *numeros*, en su árbol de directorios, y agregue el permiso de escritura a todos los usuarios del archivo. No utilice tubería.
8. Cree el archivo *procesos,* en el directorio *unidad2,* que contenga información extendida referente a los procesos activos en el sistema.
9. Realice un enlace duro al archivo *procesos* en el directorio *trabajo*, con el nombre *pro-ln*. Utilice nombre relativo. ¿Cuáles son las características a destacar de un enlace duro?
10. Realice una copia de respaldo en su directorio actual de los archivos creados en este práctico, con el nombre *copias.tar.* Y luego comprima el archivo creado.
11. Muestre el contenido del archivo comprimido *copia.tar.Z* y luego descomprímalo.
12. Cambie los permisos al archivo *pro-ln*, de forma que el dueño tenga todos los permisos, el grupo pueda leerlo y modificarlo, los otros tengan negados sus permisos (método absoluto).
13. Desde el directorio de login borre todos los archivos creados (sólo los archivos).

```
1. mkdir  trabajo  practico ; mkdir  trabajo/unidad1  trabajo/unidad2
2. ls   /dev/pt*  trabajo/unidad1/terminales
3. ls –li  /etc  >  numeros
4. grep  "^.rwx-----x"  numeros  >  practico/permisos
5. ls  -li  /etc  |  grep  "^.rwx-----x"   >  practico/permisos
6. cd  trabajo/unidad2
   ls  ../../..  |  more
7. find ~/  -name numeros –exec chmod a+w  {}  \
```

la opción *–exec* permite especificar un comando de linux que se ejecutará tomando como argumento c/u de los archivos encontrados por el comando f*ind.* Coloque {} donde se insertará el nombre del archivo y agregue \ al final del comando para completar la sintaxis. *Chmod* se ejecutará en cada archivo llamado *numeros.*

```
8. ps  -aux  > procesos
9. ln  procesos  ../pro-ln
10. tar cvf copia.tar ../unidad1/terminales  ../../practico/permisos ../../numeros
    procesos
    gzip copia.tar
11.   zcat copia.tar.gz
    gunzip copia.tar.gz
12. chmod  u=rwx,g=rw,o=   ../pro-ln
13. cd
    rm  trabajo/unidad1/*
    rm  trabajo/unidad2/*
    rm  números ; rm practico/permisos
```

LINUX: PRÁCTICO INTEGRADOR N° 2

Temas: Enlaces, permisos, administración de procesos, tuberías, redirección de entrada/salida.

1. Cree, en el directorio de conexión, el subdirectorio *parcial***.**
2. Cree el archivo *lista* en el directorio parcial, con la salida del comando *ls –l /temp*
3. Cree, en su directorio de conexión, un acceso directo (enlace simbólico) al archivo *lista*, con el nombre *list-ls.*
4. Muestre por pantalla el nro de i-nodo del archivo *lista.* Recuerde este número.
5. Ejecute el comando necesario para permitir que el archivo *lista* figure en el directorio de login o conexión con distinto n° de i-nodo.
6. Idem al anterior pero con el mismo n° de i-nodo y con el nombre de *list-1*.
7. Ejecute el comando necesario para permitir que el archivo *lista* figure en el directorio de login o conexión como *lista-2* pero con el mismo n° de i-nodo, logrando en la misma acción que el archivo *lista* desaparezca del directorio parcial.
8. Analice la diferencia entre utilizar el comando *cp* y *mv*.
9. Muestre en listado extendido, los procesos que se están ejecutando en su sesión.
10. Diga cuáles son los campos que identifican el n° de proceso y el n° del proceso padre.
11. Muestre el contenido del directorio donde Linux archiva los drivers. Realice esta tarea en forma paginada.
12. Modifique los permisos del archivo *lista* de forma tal que la ejecución de la siguiente orden le de error: *ls >> lista*
13. Diga qué sucede con los permisos del archivo *lista-2,* ¿se modifican con la acción del punto anterior?
14. Ejecute la orden: *more lista-ls* y diga por qué le da error.

.....

1. `cd`
   `mkdir parcial`
2. `ls -l /tmp >> parcial/lista`
3. `ln -s parcial/lista lista-ls`
4. `ls –i parcial/lista`
5. `cp parcial/lista lista`
6. `ln parcial/lista lista-1`
7. `mv parcial/lista lista-2`
8. Compare su análisis con sus compañeros
9. `ps -f`
10. El campo PID muestra el número de proceso, y el campo PPID muestra el id del proceso padre.
11. `ls /dev | less`
12. `chmod 444 lista`
13. No se modifican los permisos del archivo *lista-2*, pues el archivo lista, generado en el punto 5, es una copia de *parcial/lista* y tiene distinto nodo-i.
14. El archivo *lista-ls* es un enlace simbólico al archivo *parcial/lista*, que ya no existe, por lo tanto no lo encuentra.

LINUX: PRÁCTICO INTEGRADOR N° 3

Temas: Procesos, filtros, tuberías, redirección de entrada/salida, usuarios.

1. Marque la salida de la siguiente línea de comando: who >> usuarios

A	Los usuarios conectados al sistema	C	El contenido del directorio padre
B	El contenido del archivo usuarios	D	Ninguna de las anteriores

2. Idem al anterior : who | tail -5

A	Los 5 usuarios conectados al sistema dos veces	C	Las 5 últimas líneas de salida del comando who.
B	Las 5 primeras de las 5 últimas líneas de salida del comando who.	D	Ninguna de las anteriores

3. Dadas las siguientes líneas de comandos, marque la salida de la última línea de comando:

mv **..**/listado **.**

more –10 ../listado

A	Listado de los archivos que comienzan con **lis***	C	El contenido del directorio padre
B	El contenido del archivo listado	D	Mensaje de error: no encuentra el archivo

4. Marque la salida de la siguiente línea de comando: grep nd **xyz** | wc –cl

A	Listado de las líneas del archivo **xyz** que comienzan con **nd**	C	La cantidad de líneas y caracteres de cada línea que tiene la cadena **nd.**
B	Listado de las líneas del archivo **xyz** que contienen la cadena **nd**	D	La cantidad de líneas y caracteres de todas las líneas que tienen la cadena **nd**

5. Marque la salida de la siguiente línea de comando: who | tail -10 | head –5

A	Los 5 usuarios conectados al sistema dos veces	C	Las 5 últimas y las 5 primeras líneas del comando who.
B	Las 5 primeras de las 10 últimas líneas de salida del comando who.	D	Ninguna de las anteriores

6. Marque el comando correcto para mostrar información detallada sobre el proceso N° 5432 si es que está corriendo en su terminal:

A	Ps | grep 5432	C	ps -f | grep 5432
B	Ps aux >> grep 5432	D	Ninguno de los anteriores

1. D
2. C
3. D
4. D
5. B
6. C

LINUX: PRÁCTICO INTEGRADOR N° 4

Temas: Variables. Estructura condicional. Programación del shell.

1. Crear un script que permita guardar en un archivo, el nombre y n° de i-nodo de los archivos de un directorio específico, ordenado por n° de i-nodo. Utilice parámetros posicionales. Ejecute el archivo con sh o como ./nom_script.

```
If  test  $#  -eq  2 >  /dev/null
then
      if  test  -d  $1
      then
            ls  -i  $1  | sort –n –o  $2
      else
            ls  -i  $2 | sort  -n –o  $1
      fi
else
      echo  "Ingrese correctamente los parámetros"
fi
```

2. Crear un script que: a) guarde el nombre de los archivos creados en la última hora de trabajo, que se encuentren en un directorio específico. b) agregue al archivo información extendida de los procesos activos perteneciente a un usuario en particular.

```
clear
echo  -n "Ingrese el nombre del directorio:  "
read  d1
echo  -n "Ingrese el nombre del usuario: "
read  us1
find  $d1  -maxdepth  1  -amin  -60  > info
ps –aux  |  grep  $us1 >>  info
```

Ejecute el script con los siguientes directorios: /proc, /root, /rhome/alumno50, /dev. Analice la información contenida en el archivo info, luego de cada ejecución.

3. Crear un script que le permita agregar informes sobre los procesos corriendo en su terminal a un archivo específico, mientras éste exista. a) Ejecute el script en background. b) Verifique, con la orden ps –f, la existencia del proceso en background. c) Analice la información contenida en el archivo. d) Borre el archivo de informes. e) Verifique, con la orden ps –f, que el proceso en background haya terminado.

```
while  test  -a  $1
do
      ps >>  $1      # el primer parámetro posicional es el nombre del
                     # archivo donde guardamos los informes.
done
```

LINUX: PRÁCTICO INTEGRADOR N° 5

Temas: Variables. Estructura condicional. Programación del shell.

1. Crear un script de opciones, con los siguientes ítems: - Comparar directorios (muestre los archivos comunes) – Mover archivo. – Informar sobre uso de la memoria. – Informar sobre los sistemas de archivos montados. – Salir.

```
      # Script de opciones: menuop
while true
do
echo –n "Menu de opciones
      1 – Comparar directorios
      2 - Mover archivo
      3 - Informe de memoria
      4 - Informe Sist. Montados
      5 - Salir
Ingrese opción: "
read  resp
case  $resp  in
      1 | c)   echo –n " Ingrese el nombre de un directorio: "
            read d1
            echo
            echo –n "Ingrese el nombre de otro directorio: "
            read d2
            cd  $d1
            ls  –1  $d2  > dire
            for  i  in  `ls  -1`
            do
                  if  grep  $I  dire  >  /dev/null
                  then
                        echo $i "esta en ambos directorios"
                  fi
            done
            read a;;

      2 | v) echo –n "Ingrese el nombre del archivo a mover: "
               read  a1
               echo  -n "Directorio destino: "
               read  a2
               (find /  –name  $a1> /dev/null && mv $a1 $a2; echo listo) || echo
      "no se   encontro $a1"
               read a;;

      3 | i)  free  -m
                read  a;;

      4 | m)  mount
                 read  a;;
      5 | s)  exit
esac
done
```

LINUX: PRACTICO INTEGRADOR Nº 6

Tema: Programación del shell

1) Explique qué hace la línea 04 del script shell.

2) ¿Qué ve el operador en pantalla si selecciona la opción 3 del menú?

3) Explique el funcionamiento de la línea 18.

#	Instrucciones
01	#!/bin/bash
02	# Script Recuperatorio 2º – 2003 – 2k10
03	# ----------
04	PATH= **. :** $PATH ; export PATH
05	opcion=9
06	clear
07	while true
08	do
09	echo “1. Procesos en ejecución.”
10	echo “2. Fecha del día.”
11	echo “3. Espacio en disco.”
12	echo “----------“
13	echo “9. Finaliza”
14	echo “----------“
15	echo –e “Su opción: \a\c”
16	read opcion
17	case $opcion in
18	1) clear ; top ; read nada ;;
19	2) batch < script1.sh ;;
20	# El archivo script1.sh contiene (sin #):
21	# clear ;echo “Hoy es `date`” ;read nada
22	3) clear ; df ; read nada ;;
23	9) break ;;
24	*) echo –e “\n $opcion ??? \a”
25	read nada
26	;;
27	esac
28	done

4) ¿Qué significa SPOOL y para qué se lo utiliza en LINUX?

5) Explique el funcionamiento de la instrucción “for” del shell de UNIX y de un ejemplo de uso.

LINUX: PRACTICO INTEGRADOR Nº 7

Tema: Permisos, filtros, memoria, procesos, variables del shell

1. Crear un subdirectorio llamado <legajo> (su legajo). Realice el práctico en este subdirectorio.
2. Copiar el contenido del archivo **/etc/passwd** con el nombre **lista.**
3. Modifique los permisos del archivo **lista** de forma tal que la ejecución de la siguiente orden le de error: *ls* >> lista
4. Explique brevemente el modo numérico del comando chmod. Ejemplifique
5. La salida de la siguiente línea de comando muestra: (who >> usuarios ; cp usuarios ..) && ls /
   a. Los usuarios conectados al sistema
   b. El contenido del directorio padre
   c. El contenido del archivo usuarios
   d. Ninguna de las anteriores

6. El comando para verificar que el usuario **alumno12** esté conectado es:
   a. grep alumno12 who | echo "usuario alumno12 conectado"
   b. (who | grep alunmo12) && echo "usuario alumno12 conec."
   c. (who ; grep alumno12) && echo "usuario alumno12 conectado"
   d. who | write alumno12 && grep alumno12

7. El comando para buscar, en background, el archivo **alturas** en el file system mientras edita el archivo **menu** es:
   a. find / altura && vi menu
   b. find / -name alturas > ruta & ; vi menu
   c. vi menu | grep altura
   d. grep / -name alturas > ruta & ; vi menu

8. El directorio /proc:
   a. Es un sistema de archivos virtual.
   b. Sus archivos tienen 0 bytes de tamaño.
   c. Sus archivos están asociados a los dispositivos periféricos.
   d. Sus archivos están asociados a los procesos activos.

9. El comando que clasifique por número de i-nodo los archivos de su directorio de conexión, es:
   a. sort -n | ls –i ~/
   b. ls -i ~/ | sort -n
   c. find / -user $LOGNAME | sort -n
   d. ls –i | grep 160179

10. Relacione:

a	renice		Visualiza procesos que se están ejecutando en background.
b	/var/spool/lpd		Nombre por defecto del dispositivo impresora.
c	swap		Guarda el nombre de la impresora definida por defecto
d	jobs		Almacena el estado de salida del último comando ejecutado.
e	lp		Crea un sistema de archivos en un disquete.
f	Kill -9		Directorio raíz de la cola de impresión.
g	PRINTER		Archivo de bloque.
h	$?		Asigna prioridad a un proceso.
i	SHELL		Fuerza la terminación del proceso.
J	free		Área que alberga páginas de memoria RAM
K	mke2fs /dev/fd0H1440		Comunicación de procesos
L	Pipeline		Indica la ruta absoluta al programa intérprete de comandos.
M	/dev/hda2		Muestra la cantidad total de memoria física y de intercambio.

LINUX: PRACTICO INTEGRADOR Nº 8

Tema: Parámetros posicionales, manejo de archivos, procesos, programación del shell

1. Si su sesión de trabajo termina a las 20hs., deje en un archivo en su correo:
   A – un listado de los archivos que contiene en el directorio de conexión.
   B – un listado de los usuarios conectados al sistema.
   Muestre por pantalla las tareas planificadas. (Planificación de procesos)

2. Defina y ejemplifique las siguientes variables del shell:
HOME: .....
SHELL: .....
LOGNAME: .....
PATH: ……..
MAIL: …….

3. Explique el tema parámetro posicional, comente cómo los utiliza el shell.

4. Crear un shellscript que permita agregarle a todos los archivos de un directorio (sólo a los archivos) la extensión "**.c**". Utilice parámetros posicionales.

5. Crear un shellscript que controle e informe sobre el número de terminales abiertas por un usuario, estas no pueden ser más de 4.

6. Muestre un informe completo (paginado) de los procesos que están corriendo en el sistema. Explique su contenido.

7. Explique qué información tiene la variable especial **$?** y cómo la utiliza el shell.

8. Busque el archivo **examen**, a partir de su directorio padre, cuyo dueño es el usuario **alumno38**.

9. Genere un archivo de control que contenga un informe completo sobre los procesos, memoria utilizada y paginada. Realice la tarea cada 30 segundos y por el lapso de 5 minutos. Realice esta tarea en background.

LINUX: PRACTICO INTEGRADOR Nº 9

Tema: empaquetado de archivos, compresión.

Describa brevemente para qué sirven los siguientes comandos y sus opciones.

1. $ tar -tvf archivo.tar

2. tar -xvf archivo.tar

3. tar -cvf archivo.tar A*

4. gzip -9 archivo.tar

5. gzip -d archivo.tar.gz

6. tar -tzf archivo.tar.gz

Apéndice I

Metacaracteres

En Linux, los comandos utilizan una serie de caracteres especiales o comodines, que en combinación con literales alfanuméricos, permiten manejar varios archivos utilizando, en la línea de comandos, secuencias de escritura muy cortas. El usuario puede introducir en cualquier comando uno o varios patrones, es decir, una palabra que contiene uno o varios comodines, y el shell sustituirá cada patrón por una lista de todos los nombres de archivos que coinciden con ese patrón. Los comandos nunca ven los metacaracteres o comodines.

Los nombres de los archivos pueden expresarse de una manera genérica de tal manera que definan un cierto tipo de archivo (basado en los nombres de los archivos), más bien que referirse a nombres particulares.

A continuación se presentan los metacaracteres de mayor uso:

*	Coincide con ninguno o con varios caracteres
?	Coincide con cualquier carácter individual
[ccc]	Coincide con cualquiera de las alternativas de caracteres entre corchetes. Son legales los intervalos [0-9] o [a-z]
[!ccc]	Coincide con cualquier carácter excepto aquellos entre corchetes.
;	Termina comando: *comando1; comando2*
\	Toma literalmente el carácter que le sigue
'...'	Toma literalmente el conjunto de caracteres entre las comillas
"..."	Toma literalmente el conjunto de caracteres entre las comillas, excepto $, \, `...`
`...`	Ejecuta el comando que está entrecomillado
#	Indica que el conjunto de caracteres siguiente, es un comentario

El comodín asterisco *

El asterisco es uno de los comodines más usados ya que coincide con cero o todos los caracteres. Es decir que el patrón **S.*z** podría corresponder a la cadena **S.z**, **S.1z**, **S.1234z**, etc.

Ejemplo 1

```
$ ls docu.*
```

En este caso, el comando *ls* entrega un listado de todos los archivos cuyo nombre sea *docu,* cualquiera sea su extensión.

Ejemplo 2

```
$ ls *.txt
```

Aquí el comando *ls* entrega una lista de todos los archivos cuya extensión sea **.txt**.

El comodín ?

Este comodín reemplaza a cualquier carácter individual que esté en la posición, dentro de la cadena, donde se encuentra el comodín.

Ejemplo 1

```
$ ls prog?.c
```

Este comando lista todos los archivos del directorio actual que comiencen con “prog” y tengan extensión “.c”, tal como prog1.c, prog2.c, progA.c, progB.c, etc.

Ejemplo 2

```
$ cp dato? ..
```

En este ejemplo, el shell crea la lista en orden alfabético de todos los archivos que encuentre en el directorio actual, cuyo nombre sea por ejemplo *dato1, datoc, dato5, datof,* etc, y recién entonces la pasa a ***cp***. El comando cp nunca ve el **?**. La búsqueda que realiza el shell en el directorio actual genera una lista de cadenas que se pasa al comando ***cp***, el comando *cp* copia la lista de archivos al directorio padre. Es decir, que el comodín **?**, reemplaza a cualquier carácter que esté en esa posición.

Ejemplo 3

```
$ cp dat?? ..
```

En este caso, se copian todos los archivos que se encuentren en el directorio actual, cuyo nombre sea por ejemplo *dato1, dat2c, dat35, datff,* etc, al directorio padre. Es decir, que en los últimos dos caracteres del nombre de los archivos que comienzan con *dat,* puede haber cualquier combinación de caracteres alfanuméricos.

Rangos [abc] [a-z] [0-9]

Como primer uso, podemos decir que los corchetes permiten delimitar a un conjunto de caracteres, de los cuales al menos uno debe coincidir con el carácter de la cadena, que esté en una posición específica.

Ejemplo 1

Suponiendo que necesitamos copiar el archivo *eje* al directorio padre, sabemos que lo identificamos con una letra al final del nombre, pero no recordamos si es **a**, **f** o **z,** por lo cual podríamos tipear:

```
$ cp eje[afz] ..
```

Así copiará cualquier archivo cuyo nombre comience con la cadena eje y finalice con a, f o z, es decir que puede copiar el archivo *ejea, ejef* y *ejez.*

Como segundo uso podemos especificar dentro de los corchetes un intervalo, ya sea alfabético o numérico, de valores que puede asumir un determinado carácter en una cadena que utilizamos como patrón.

Ejemplo 2

```
$ ls -l r[1-5]ta
```

El comando anterior realizará un listado extendido de los archivos r1ta, r2ta, r3ta, r4ta y r5ta.

Comillas simples '...'

Toma literalmente el conjunto de caracteres entre las comillas, es decir que no interpreta los caracteres especiales que se hayan incluido en la cadena delimitada por las comillas simples.

Ejemplo

```
$ echo 'Las rutas de búsqueda son: $PATH'

Las rutas de búsqueda son: $PATH
```

El comando *echo* repitió la cadena de caracteres ingresada, sin que el shell haga alguna interpretación.

Comillas dobles "..."

Toma literalmente el conjunto de caracteres entre las comillas, excepto los caracteres $, \, `…`, es decir, que permite que el shell los interprete.

Ejemplo 1

```
$ echo "Las rutas de búsqueda son: $PATH"
Las rutas de búsqueda son: /bin:/sbin:/usr/bin
```

El comando *echo* muestra la cadena de caracteres ingresada, y el contenido de la variable PATH, es decir las rutas absolutas a los directorios donde están los archivos ejecutables.

Ejemplo 2

```
$ echo "La fecha es: `date`"
La fecha es: 06-23-18
```

El comando echo nos muestra la cadena ingresada y la salida del comando *date* en el formato configurado en su sistema.

Signo

Este comodín se utiliza cuando en un archivo ejecutable, se desea introducir un comentario o aclaración. Todo lo escrito a partir del signo # no será ejecutado por el Shell.

Ejemplo

```
$ cat cuenta_archivos
# este archivo muestra la cantidad de archivos en el directorio actual
# no se distinguen archivos de directorios
echo "Hay `ls | wc –w` archivos en el directorio actual
```

En el ejemplo anterior se muestra un archivo que contabiliza la cantidad de archivos que se encuentran en el directorio en donde se ejecute. El signo # se utiliza simplemente, para hacer comentarios acerca del archivo.

Apéndice II

Introducción al Editor de Textos VI

Un editor de texto nos permite crear y modificar archivos de texto que contiene palabras y caracteres que fueron escritos desde el teclado, pero no dispone de los caracteres especiales del idioma castellano (acentos, eñes, etc.). No debemos confundir un editor de textos con un procesador de texto, este último no sólo permite editar textos sino que también nos da la posibilidad de modificar la presentación del mismo a través de la selección del tipo de fuente y estilo.

En Linux todo usuario debe conocer los comandos básicos para editar archivos con ***vi,*** ya que es un editor que siempre está disponible en todos los sistemas Unix, y tiene una ventaja muy importante, y es que permite desde él mismo invocar todas los comandos del shell, otorgando gran flexibilidad al depurar programas y ejecutarlos.

Este apéndice tiene como objetivo mostrar las bases para utilizar el editor ***vi*** y editar cualquier archivo.

¿Cómo iniciar *vi*?

Para utilizar este editor de textos sólo es necesario especificar el nombre de archivo a editar, indicando la ruta de acceso al mismo, si el archivo se encuentra en otro subdirectorio. Si el archivo es nuevo y aún no existe, **vi** lo creará en ese momento.

Sintaxis:

```
$ vi nombre_de_archivo
```

La pantalla se despejará en ese momento y mostrará el contenido del archivo. Teniendo en cuenta una pantalla típica de 24 líneas, la estructura de la pantalla con **vi** tendrá las siguientes secciones:

- Líneas de contenido del archivo
- Líneas con un guión (~) indicando líneas sin texto, que solo aparecerán en caso que el texto no supere la cantidad de líneas máximas de la pantalla.
- Una línea en la parte inferior para introducir comandos **vi**, que se muestra con dos puntos (**:**).

Ejemplo

```
$ vi  cumpleaños
Luciana  12 de Mayo
Pablo  25 de Abril
Julieta 3 de Noviembre
Ariel   12 de Julio
~
~
```

En este archivo tenemos 4 líneas de texto, la cuarta línea es el fin del archivo y las 19 líneas restantes no poseen texto y se muestran con guión.

Si estamos creando un archivo todas las líneas comenzaran con (~) lo que indica que son líneas más allá del fin de archivo. Si por el contrario el archivo ya existe solo mostrará con (~) las líneas que no posean texto o caracteres. Pero siempre como última, una línea libre para introducir comandos.

Ejemplo 1

```
$ vi archivo1  archivo2
```

Si se pasan dos archivos como parámetros, se edita primero el archivo1 y luego el archivo2.

Ejemplo 2

```
$ vi +$ archivo
```

Este comando edita el archivo y nos posiciona al final del mismo.

Ejemplo 3

```
$ vi +18 archivo
```

Se puede abrir el archivo y posicionarnos directamente en la línea especificada, por ejemplo en la línea 18

Ejemplo 4

```
$ vi +/ejemplo archivo
```

Este comando edita el archivo y coloca el cursor en la primera posición de la palabra "ejemplo"

Editar un archivo con el editor **vi** nos presenta tres posibilidades de trabajo, a esto se lo denomina Modos y son tres: modo inserción, modo comando y modo lectura.

Modo Inserción

Este modo nos permite escribir un texto. Se debe introducir ***i*** o ***a*** para activar el modo *Inserción*, en el cual introduciremos el texto. Las opciones disponibles son:

i todo texto se introduce a la izquierda de la posición de cursor.
I el texto se inserta al principio de la línea.
a todo texto se inserta a la derecha de la posición del cursor.
A añade texto al final de una línea.
o inserta una línea en blanco debajo de la línea activa.
O inserta una línea en blanco arriba de la línea activa.
R sobre escribe el texto.

En este modo se va visualizando lo que se escribe y se pulsa **Esc** para desactivar el modo Inserción. De esta manera se vuelve al modo Comando.

Modo Comando

En este modo los caracteres ingresados son interpretados por el ***vi*** como comandos, es decir que cada letra o caracter que se escribe, es un comando ***vi,*** que hace que ocurra alguna acción, por lo tanto lo que se escribe no se ve en pantalla.

Comandos para moverse o desplazarse con el cursor:

h mueve el cursor a la izquierda
j mueve el cursor hacia abajo
k mueve el cursor hacia arriba
l mueve el cursor hacia la derecha
0 (cero) mueve el cursor al principio de la línea
$ mueve el cursor al final de la línea

Cuando se introducen estos comandos no aparecen en la pantalla y el cursor se desplazará en la dirección deseada.

Nota: *se sugiere no depender demasiado de la teclas de flecha (←↑→↓), a veces se puede dar el caso de que no funcionen, ya que requieren ciertas configuraciones del teclado, del terminal y de determinados archivos del sistema, o suelen ser muy lentos en un sistema de red muy cargado*

Comandos para borrar caracteres, palabras o líneas:

x borra un carácter donde está posicionado el cursor
dd borra la línea actual
5dd borra 5 líneas a partir de donde se encuentra posicionado el cursor
D desde el cursor hasta el final de la línea
dw borra desde el cursor hasta el final de la palabra
d$ desde el cursor hasta el final de la línea
d0 borra desde el cursor al principio de la línea
m>> desplaza **m** líneas hacia la derecha (como si se tecleara <TAB>)
m<< desplaza **m** líneas hacia la izquierda

Comandos para copiar y pegar caracteres, palabras o líneas:

Y permite copiar la línea
yy permite copiar la línea
P pega lo seleccionado antes del cursor
p pega lo seleccionado después del cursor
yw copia la palabra
y$ copia desde el cursor hasta el final de la línea

Juntar líneas usando J

A diferencia de otros editores, ***vi*** no tiene carácter de final de línea que se borre para juntar dos líneas, por lo que si desea hacer ésto, sólo debe utilizar el comando **J,** ubicando el cursor a partir de la línea que desee juntar.

Deshacer cambios con el comando u

Vi recuerda la última inserción, eliminación o modificación que se hizo. Si queremos deshacerla, se debe introducir el comando **u**. No es necesario estar cerca de la línea donde se hizo

el cambio cuando ejecuta el comando **u**. Los comandos para desplazar el cursor son ignorados por **u** por lo que no se pueden deshacer.

El editor vi en su modo comando también permite el acceso a otros comandos adicionales y más complejos. Estando en el modo Inserción para ingresar debemos presionar la tecla Esc y luego escribimos dos puntos "**:**". Estando en el modo comando solo escribimos dos puntos "**:**". Algunas de las opciones son:

Ver el nombre del archivo, el tamaño y su ubicación

:f muestra el archivo que se está editando, en qué línea se encuentra el cursor y cuántas líneas tiene. Ctrl+G es una forma alternativa de mostrar la misma información que **:f**.

Ejemplo

```
:f
Trabajo    [Modified]   line 20 of 135   - - 32% - -  col 1
```

Guardar el trabajo

Cuando trabajamos con **vi**, el contenido de los archivos que se está editando queda almacenado en la memoria RAM y es preciso guardar las modificaciones realizadas para que los cambios tengan efecto. Para ello se utiliza el comando **w** para guardar el trabajo actual en un archivo en disco de la siguiente forma:

```
:w archivo
```

Si está realizando muchos cambios en una sesión vi, es prudente guardar el archivo periódicamente. En ese caso se introduce sólo **:w**, pero si desea guardar el trabajo en un archivo diferente al que se está editando debe asignarle un nombre nuevo:

```
:w archivo_nuevo
```

Cómo salir de vi y/o anular los cambios

:wq graba el archivo y sale del editor de textos
ZZ graba el archivo y sale del editor de textos
:q! sale del editor sin tener en cuenta los cambios realizados. Vuelve al indicador del shell

Cómo ejecutar comandos del Shell desde el editor de textos vi sin salir del editor

Para ello, se utiliza el signo de admiración "**:!**"

:!comando ejecuta el comando del Shell seleccionado

Ejemplo

Para ver el contenido del directorio */bin*:

```
:!ls /bin
```

Esto permitirá que ejecute cualquier comando externo de Linux sin salir del editor

Comandos de búsqueda de textos

:/texto busca un texto hacia delante, es decir a partir de donde se encuentra posicionado el cursor, hasta el final del archivo
:?texto busca un texto hacia atrás, o sea desde la posición del cursor al comienzo del archivo

Comandos de sustitución de cadenas

A continuación de los dos puntos (**:**) podremos ejecutar comandos de búsqueda o sustitución de cadenas, de la siguiente forma:

```
:1,$ s/cadena1/cadena2
```

Este comando sustituye la cadena1 por la cadena2 sólo en la primera ocurrencia

```
:1,$ s/cadena1/cadena2/g
```

Con la opción **g** agregada a la anterior, realizará la sustitución de la cadena1 por la cadena2 en todas las ocurrencias que se produzcan en el archivo.

Para completar cualquier comando que comience con dos puntos (:) pulse Enter

Modo solo Lectura

Es una buena precaución en aquellos casos en que solo se necesita ver el contenido de un archivo y no se quiere realizar ningún cambio, editar un archivo en modo ***Solo Lectura.*** Para ello, sustituya el comando vi por view en la línea de comandos de la siguiente forma

```
$ view archivo
```

View inicia el editor vi en modo solo lectura, y si intenta escribir el archivo presentará un mensaje de error.

```
Warning: Changing a readonly file (cambiando un archivo de sólo lectura)
```

Se puede anular el modo Solo Lectura de view y guardar los cambios, de la siguiente manera: **:w!**

Esto provoca que se abandone el *Modo Solo Lectura* y se ingrese al *Modo Comando* donde el editor nos permite modificar el archivo

REFERENCIAS

- Actualizaciones del kernel https://www.linux.org/threads/kernel-updates.18152/
- Administrador de sistemas y servicios https://www.freedesktop.org/wiki/Software/systemd/
- Administración de Red Hat Linux al descubierto - Thomas Schenk et al – (Edit. Prentice Hall - 2001)
- Aprendiendo Linux - Guía en 10 minutos John Ray - (Edit. Prentice Hall - 2000)
- Apunte – Curso: Linux a fondo - Oscar Espeche – (Extensión Universitaria- UTN FRC – 2002)
- Árbol de directorios https://www.debian.org/releases/stable/s390x/apcs02.html.es
- Btrfs Main Page https://btrfs.wiki.kernel.org/index.php/Main_Page
- Cómo Trabajar con Unix - Remiro del Caz - (Edición - 1998)
- Compumagazine Linux- Manual de Referencia - Luis Tomas Wayar – (PC Forum SA - 1999)
- Debian https://www.debian.org/doc/user-manuals#securing
- Debian https://wiki.debian.org/es
- El Entorno de programación Unix - Brian W. Kernighan - Rob Pike – (Edit. Prentice Hall - 1987)
- El Sistema Operativo Unix. Introducción y aplicaciones - José Canosa – (Edit. Marcombo – 1988)
- El Sistema Operativo GNU https://www.gnu.org/home.es.html
- El Sistema Operativo GNU-Software https://www.gnu.org/software/software.es.html
- El Sistema Operativo GNU-Software https://www.gnu.org/software/coreutils/manual/html_node
- Entendiendo el kernel de Linux https://www.linux.org/threads/understanding-the-linux-kernel.12508/
- Expresión regular https://www.linux.org/threads/lfcs-regular-expression-part-2.4558/
- Frontpage Debian Wiki https://wiki.debian.org/es
- Fundamentos de Linux https://help.ubuntu.com/kubuntu/desktopguide/es/linux-basics.html
- GNU Manuals Online – GNU Project - Free Software Foundation
- https://www.gnu.org/manual/manual.html

- GRUB Manual http://www.gnu.org/software/grub/manual/grub/html_node/Overview.html#Overview
- KernelNewbies: Ext4 https://kernelnewbies.org/Ext4#head-38e6ac2b5f58f10989d72386e6f9cc2ef7217fb0
- La Biblia de Red Hat Linux 6 - Arman Danesh – (Edit. ANAYA - 1999)
- Linux - Jack Tackett Jr – Steve Burnett. (Edit. Prentice Hall – 1999)
- Linux – Instalación, administración y uso del sistema - Vicente J. Blanco – (Edit. Rama – 1996)
- Linux- Suse 6.3. Instalación, Configuración y primeros pasos - Suse GmbH (Edit. Suse - 1999)
- Linux https://www.linux.org/docs/
- **Linux y el sistema GNU** https://www.gnu.org/gnu/linux-and-gnu.es.html
- Linx man pages at Linux.Org https://www.linux.org/docs/man5
- **Linux Standard Base Core Specification 3.2** https://refspecs.linuxfoundation.org/LSB_3.2.0/LSB-Core-generic/LSB-Core-generic.html
- Manual de ayuda en línea de Red Hat Linux v 6.2
- Manual http://manpages.debian.org
- Nucleo Linux https://lkml.org/lkml/2011/5/29/204
- Programación en Linux, con ejemplos - Kurt Wall (Edit. Prentice Hall - 2000)
- Red Hat 9 - Guía de Referencia y Guía del Usuario en Internet - 2003
- Red Hat Linux 6.1. - Juan Carlos Espinosa – (Edit. Alfaomega-Marcombo - 2000)
- SDB: Fundamentos del sistema de ficheros https://es.opensuse.org/SDB:Fundamentos_del_sistema_de_ficheros
- Serie Práctica Unix - Steve Moritsugu y DTR Business System, Inc. – (Edit. Prentice Hall - 2000)
- Sistemas Operativos - Willian Stallings - (Edit. Prentice Hall - 1997)
- Sistemas Operativos Modernos Andrew S. Tanenbaum – (Edit. Prentice Hall – 1993)
- Sitio oficial de GNU GRUB http://www.gnu.org/software/grub/index.html
- Software- Proyecto GNU Free Software Fundation https://www.gnu.org/software
- The Linux Documentation Project https://www.tldp.org/

Los autores

Sandra Liliana Allende

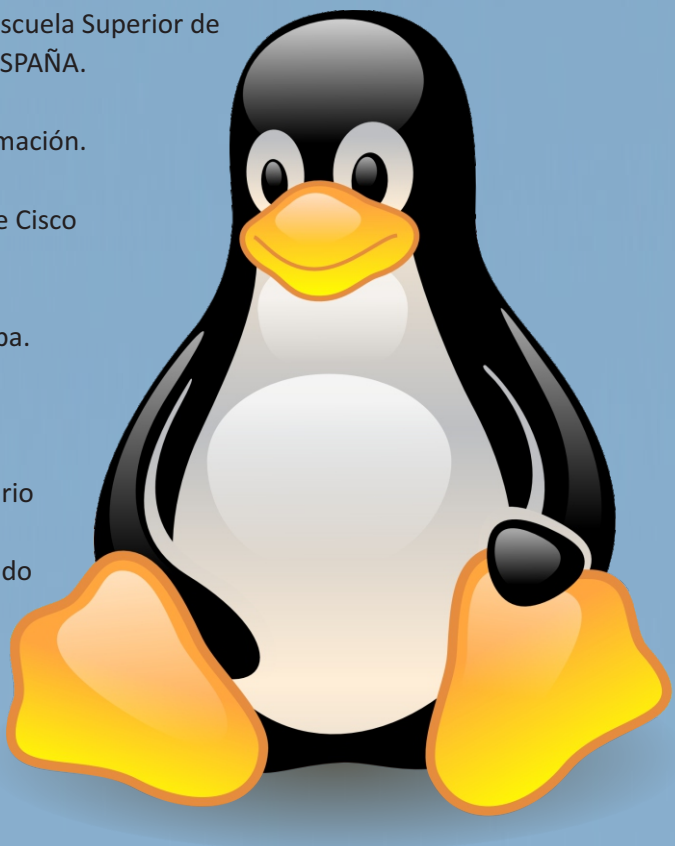
- Ingeniera en Sistemas de Información, UTN - Facultad Regional Córdoba.
- Especialista en Docencia Universitaria.
- Jefe de Trabajos Prácticos Ordinario de Sistemas Operativos y de Sistemas y Organizaciones.
- Docente Investigador Categoría V en Programa de Incentivos - Ministerio de Ciencia y Tecnología.
- Docente en la Universidad Nacional de La Rioja, Sede Chamical, también en la Universidad Atlántida Argentina, Mar del Plata y en Institutos de Nivel Terciario de la ciudad de Córdoba.

Fabian Alejandro Gibellini

- Ingeniero en Sistemas de Información, UTN - Facultad Regional Córdoba.
- Responsable del Laboratorio de Ingeniería en Sistemas de Información UTN - FRC.
- Director de Proyectos de I+D+i, en el área de Seguridad en Sistemas de Información y Software.
- Docente adjunto en las cátedras de Sistemas Operativos y Matemáticas Discretas y jefe de trabajos Prácticos de Redes de Información.
- Consejero Departamental de la carrera de Ingeniería en Sistemas de Información.
- Categorizado D en la Carrera de Docente Investigador de la UTN - Orientación Ciencias de la Ingeniería y Tecnológicas, y Categorizado IV en Programa de Incentivos - Ministerio de Ciencia y Tecnología.
- Consultor y Auditor informático.
- Instructor Certificado de la Academia Cisco.

Cecilia Beatriz Sánchez

- Ingeniera en Sistemas de Información. UTN, Facultad Regional Córdoba.
- Master en Sistemas y Redes de Comunicaciones, Opción Telemática. Escuela Superior de Ingenieros en Telecomunicación. Universidad Politécnica de Madrid. ESPAÑA.
- Especialista en Docencia Universitaria. UTN FRC.
- Profesor Titular cátedra Sistemas Operativos y cátedra Redes de Información.
- Coordinadora de la Cátedra Comunicaciones y Redes
- Instructora Certificada del Programa de Networking de la Academia de Cisco



Monica Mariel Serna

- Ingeniera en Sistemas de Información, UTN - Facultad Regional Córdoba.
- Jefe de Trabajos Prácticos Ordinario de Sistemas Operativos.
- Jefe de Trabajos Prácticos Ordinario de Arquitectura e Computadores.
- Profesor Adjunto interino de Matemática Discreta.
- Docente Investigador Categoría V en Programa de Incentivos - Ministerio de Ciencia y Tecnología.
- Profesor Titular interino de Investigación Operativa I y Profesor Asociado ordinario de Investigación Operativa II en la Universidad Nacional de La Rioja, Sede Chamical

CiN REUN Red de Editoriales de Universidades Nacionales de la Argentina
LUA Libro Universitario Argentino
edUTecNe