



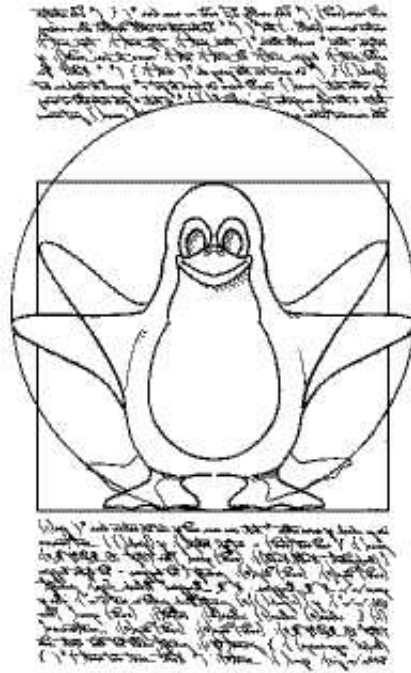
Universidad Técnica Federico Santa María



Curso Linux Básico

Departamento Informática U.T.F.S.M.

Luis Eduardo Arévalo Reyes
larevalo@inf.utfsm.cl



VALPARAÍSO, JULIO 2004

Índice

1. Introducción	3
1.1. Agradecimientos	3
1.2. ¿ Por que Linux?	3
2. Sistemas Operativos & GNU/Linux	5
2.1. Componentes de un Sistema Computacional	5
2.1.1. Perspectiva Tecnológica	5
2.1.2. Perspectiva Abstracta	5
2.1.3. Funciones de un Sistema Computacional	5
2.2. Definición y funciones de un Sistema Operativo	6
2.2.1. Definición	6
2.2.2. Funcionalidades deseables en un Sistema Operativo	6
2.3. Administración de CPU y Memoria	7
2.3.1. Gestión de procesos	8
2.3.2. Gestión de Memoria Principal	9
2.4. Filosofía de trabajo en Sistemas *NIX	9
2.4.1. Filosofía Unix	9
2.4.2. GNU y Linux	10
3. Shell y Herramientas	12
3.1. Kit de supervivencia en la línea de comandos	12
3.1.1. El intérprete de comandos <code>bash(1)</code>	12
3.1.2. Sistemas de ayuda	13
3.1.3. Comandos Básicos	15
3.2. Conceptos Avanzados	20
3.2.1. Comandos y pipelines	20
3.2.2. Interpretación de líneas	21
3.2.3. Estructuras de control	22
3.2.4. Entrada y salida	23
3.2.5. Scripts	23
3.2.6. Configuración de la cuenta	24
3.3. Editores	24
3.3.1. El editor <code>emacs(1)</code>	24
3.3.2. El Editor <code>vi(1)</code>	24
3.3.3. Otros	26
4. GNU/Linux por dentro	27

4.1.	Bootloaders	27
4.1.1.	Configuración de Bootloader: grub y lilo	27
4.2.	Niveles de ejecución	28
4.3.	Scripts de arranque y procesos	28
4.4.	El kernel en Linux	30
4.4.1.	Comandos de manejo de módulos	30
4.4.2.	El sistema de archivos inicial	31
4.5.	Gestión Sistemas de Archivos	31
4.5.1.	Sistema de archivo de Linux	31
4.5.2.	Permisos de archivos y directorios	33
4.5.3.	Montaje y desmontaje de sistemas de archivos	35
4.5.4.	Sistemas de archivos en red	38
4.5.5.	Verificación de sistema de archivo	39
4.5.6.	Gestión de Swap	40
5.	Administración del Sistema	41
5.1.	Gestión de Usuarios y Grupos	41
5.1.1.	Perfiles de usuario	41
5.1.2.	Herramientas para la gestión de usuarios	41
5.1.3.	Grupos de usuario	43
5.2.	Administración de Paquetes	43
5.2.1.	RedHat Package Manager (RPM)	43
5.2.2.	Actualización del sistema	44
6.	Software de Productividad	47
6.1.	Entornos de Escritorio	47
6.1.1.	XFree86(1)	47
6.1.2.	Descripción de Entornos de Escritorios	48
6.2.	Aplicaciones de Oficinas	50
6.2.1.	Aplicaciones de KDE	50
6.2.2.	OpenOffice	50
6.2.3.	Aplicaciones GNU en general	50
6.3.	Herramientas de Mensajería, E-mail y Web	50
6.3.1.	Mensajería	50
6.3.2.	E-mail	51
6.3.3.	Web	53
6.4.	Juegos	53
6.4.1.	Gnome Games	54

6.4.2. KDE Games	54
6.4.3. Tux Racer	54
6.4.4. Juegos Gratuitos	54
6.4.5. Juegos Comerciales	54

1. Introducción

1.1. Agradecimientos

Este documento, es fruto del esfuerzo de varias personas que han colaborado en su desarrollo a lo largo de los últimos 3 años, durante los cuales se ha desarrollado estos cursos en el departamento de Informática de la Universidad Técnica Federico Santa María. La mayoría de ellos estudiantes y apoyados por el incentivo constante del profesor Horst von Brand y el espacio de trabajo que les ha dado el Laboratorio de Computación *LabComp*, ha permitido que el movimiento GNU/Linux haya crecido de muy buena manera.

Se hacen especiales agradecimientos a todos aquellos que participaron en la confección de este documento:

- Dr. Horst von Brand,
- Carlos Molina Ramírez,
- Marcelo Olguín Mena,
- Verónica Ramírez Duarte,
- Mauricio Vergara Ereche,
- Mauricio Araya López,
- Luis Arévalo Reyes,
- Roberto Bonvallet Carrasco.

El Objetivo de este documento es informar y enseñar de la mejor manera posible a todos aquellos que estén dando sus primeros pasos en el mundo del pingüino. Este documento no tiene ningún fin comercial ni lucrativo, por lo que este no podrá ser objeto de copia o reproducción, para venta. Es así el deseo de sus autores.

1.2. ¿ Por que Linux?

El sistema operativo **Linux**¹ nace en 1991, cuando un estudiante de la Universidad de Helsinki, en Finlandia, decide escribir su propio sistema operativo para aprender el lenguaje de máquina de su nuevo PC i386. Pronto Linus Torvalds acudió a la comunidad en Internet para que le ayudaran a desarrollar este hobby. Originalmente quería ponerle **Freeix** a su creación, dado que la idea era un sistema libre (*free*) compatible con UNIX². Cuenta la leyenda que el encargado del sitio FTP desde donde se distribuiría el sistema encontró horrible este nombre, y lo rebautizó **Linux**, en honor a su creador.

En rigor, **Linux** es el nombre del núcleo del sistema operativo (la parte que administra directamente los dispositivos de la máquina), no es un sistema operativo completo (que incluye programas para un amplio rango de aplicaciones, desde herramientas de configuración hasta programas orientados directamente al usuario, como son juegos). Se han creado muchas *distribuciones*, que toman una versión del núcleo, agregan programas de todo tipo para crear un sistema operativo completo, y empaquetan todo junto con algún sistema de instalación propio. Los programas que conforman las distribuciones provienen de múltiples fuentes independientes, más que nada programas desarrollados para UNIX que se distribuyen gratuitamente.

El núcleo (que es lo que propiamente se llama **Linux**) se ha seguido desarrollando bajo la dirección de Linus Torvalds, con el aporte de miles de entusiastas que se coordinan a través de Internet. El desarrollo ha sido vertiginoso, de un sistema que según su creador “jamás correrá en algo que no sea un PC i386

¹Linux es marca registrada de Linus Torvalds

²UNIX es marca registrada de The Open Group

con disco IDE” hoy día corre en una docena de plataformas que van desde agendas electrónicas hasta supercomputadores, y maneja una increíble variedad de dispositivos de todo tipo. De juguete se ha transformado en el sistema operativo que más rápidamente crece en el área de servidores. Hoy cuenta con el apoyo de empresas como IBM, Sun Microsystems, Hewlett-Packard, y Oracle. El objetivo del núcleo es ofrecer un sistema básicamente compatible con POSIX, el estándar en el área de sistemas operativos desarrollado por IEEE, al cual adhieren los sistemas UNIX.

Parte importante del éxito de Linux se debe al peculiar modelo de desarrollo que introdujo Linus Torvalds, quien lo describe como “release early, release often,” vale decir, “libere temprano, libere frecuentemente.” La idea es no esperar a tener una versión relativamente terminada para darla a conocer, sino distribuir versiones incipientes a la comunidad, obteniendo de parte de ella críticas y aportes que permitan completarla. Al desarrollar así en público se obtienen aportes de muchas más personas que las que podrían formar un grupo de desarrollo cerrado, personas que traen una diversidad de aptitudes y conocimientos especializados que difícilmente podrían reunirse de otra forma. Este modelo de desarrollo atrae a muchos interesados, quienes pueden aportar incluso simplemente revisando las sugerencias que se discuten. Además, este esquema de trabajo descentralizado da la oportunidad de elegir entre soluciones alternativas desarrolladas más o menos completamente en forma independiente, con lo que se obtienen soluciones mejores a la larga. Muchos de los desarrolladores del núcleo son fanáticos de la eficiencia, lo que junto con la flexibilidad que da el distribuir código fuente que se configura a la medida de ser necesario, da como resultado buen rendimiento incluso en máquinas limitadas. Por otro lado, es importante mantener código simple y limpio en el núcleo, de lo contrario será difícil integrar al proceso de desarrollo a nuevos interesados. Al estar el núcleo disponible en su totalidad en código fuente en un solo paquete, no es necesario mantener interfaces internas rígidas para acomodar versiones antiguas de partes desarrolladas por terceros (en otros sistemas, los manejadores de dispositivos). Con regularidad las interfaces internas sufren cambios importantes al aparecer nuevas ideas o en reacción a nuevas tecnologías de hardware (como la proliferación reciente de dispositivos que se pueden conectar y desconectar en caliente).

En términos coloquiales también se llama Linux a lo que en rigor son *distribuciones*, vale decir, un núcleo junto con colecciones de programas desarrollados independientemente. De cierta forma, Linux dió vida e impulso a una variedad de proyectos independientes que desarrollaban software de distribución gratuita para UNIX. Mucho del software que conforma una distribución de Linux proviene del proyecto GNU (siglas de “GNU, Not Unix”), fundado en 1986 por Richard M. Stallman con el objetivo de crear un sistema operativo de distribución libre similar al sistema UNIX propietario. Otras piezas (particularmente servidores) son las versiones de referencia desarrolladas para diversos servicios de red que se distribuyen gratuitamente. La base del ambiente gráfico proviene de desarrollos de un sistema gráfico portable y capaz de funcionar en red efectuados por un consorcio que incluía al MIT y a Digital. En forma creciente, hay desarrollo de software centrado en Linux, que también está migrando hacia otras plataformas, como son los UNIX comerciales (es el caso del ambiente gráfico Gnome, que está como alternativa en las últimas versiones del sistema operativo Solaris de Sun Microsystems) e incluso Windows (hay versiones del servidor web apache que corren en esta plataforma).

2. Sistemas Operativos & GNU/Linux

2.1. Componentes de un Sistema Computacional

Un sistema computacional es un conjunto de componentes, debidamente relacionados, que permite al hombre realizar tareas complejas de cómputo. Hoy en día, encontramos sistemas computacionales enormes y visibles como las grandes redes corporativas, o los centros de cómputo de investigación. Por otro lado, convivimos con sistemas computacionales todos los días: en el cajero automático, en el supermercado, con nuestro computador personal, etc.

2.1.1. Perspectiva Tecnológica

Si adoptamos una perspectiva practica y concreta, podemos distinguir ciertos componentes que son muy comunes en un sistema computacional.

1. Computador.
Es el elemento primordial de un sistema computacional. No se refiere particularmente al computador personal que utilizamos a diario, sino que cualquier dispositivo con capacidad de cómputo.
2. Dispositivos de Entrada/Salida.
Dispositivos para interactuar con el usuario. Reciben datos y entregan información (i.e. monitor, teclado, lector de código de barra, etc.).
3. Redes.
Enlaces de comunicación entre computadores autónomos o dispositivos. Existen muchos tipos y se dividen dependiendo de la tecnología y topología utilizada. Componente esencial de los sistemas computacionales distribuidos.

2.1.2. Perspectiva Abstracta

Más importante aun, es distinguir cuales son los componentes presentes en todo sistema computacional a nivel conceptual. Generalmente se habla de 3 capas de abstracción:

1. Hardware.
Conjunto de elementos físicos de toda índole, que actúan en el sistema computacional (i.e. procesador, disco duro, cables de red, hub, monitor, etc).
2. Software del Sistema.
Núcleo del sistema, herramientas y servicios. A este conjunto es lo que llamaremos generalmente Sistema Operativo. El usuario no suele interactuar directamente con esta capa.
3. Programas de Aplicación.
Conjunto de Software diseñado para el usuario final. Entre estos encontramos los procesadores de texto, administrador de archivos, software de producción, juegos, etc.

2.1.3. Funciones de un Sistema Computacional

Las funciones básicas de un Sistema Computacional se pueden dividir en 3 grandes áreas:

1. Gestión de Información.
Tareas de Almacenamiento, Creación de Referencias, Recuperación de Información y Comunicación.
2. Desarrollo de Software.
Programación y depuración de programas.

3. Ejecución de Programas.
Correr las aplicaciones que necesitamos.

2.2. Definición y funciones de un Sistema Operativo

Es un tanto difícil dar una definición exacta de un Sistema Operativo, ya que existen múltiples visiones sobre su naturaleza. Sin embargo, utilizaremos como punto de partida la definición funcional de S.O.

2.2.1. Definición

En esencia, un **Sistema Operativo** es un conjunto de programas que sirven de intermediario entre el Hardware y el Usuario. Suele llamarse Sistema Operativo solo al Núcleo, pero en rigor todo el Software de Sistema es lo que identificará a un sistema de otro.

- El **propósito** de un S.O. es entregar un ambiente de ejecución, es decir darle a nuestro sistema computacional, la capacidad de ejecutar las aplicaciones que el usuario desee.
- Su **meta principal** es hacer del sistema computacional algo útil y conveniente de usar.
- Su **meta secundaria** es utilizar el hardware de forma eficiente y coherente.

2.2.2. Funcionalidades deseables en un Sistema Operativo

Para cumplir su propósito y metas, el S.O debe proveer al usuario de la funcionalidad necesaria para que realice sus tareas. De la implementación acertada de estas funciones dependerá la calidad y utilidad del S.O.

- **Interfaz Usuaría.**
Debe existir un *lenguaje de comunicación* entre el sistema y el usuario. Debe ser suficientemente amigable para que el usuario pueda entenderlo y a la vez funcional para explotar las capacidades del Sistema Computacional. Se utilizan generalmente “interpretes de comandos” y/o “Sistema de Ventanas”.
- **Recursos Virtuales.**
Son abstracciones virtuales de los componentes de hardware. Suelen ser conceptos del mundo real, aplicados a nuestro Sistema Computacional para dar consistencia al mundo virtual que plantea el S.O. Este principio, ayuda a un mejor aprovechamiento de los recursos reales complejos y a una visión más entendible para el usuario final. El ejemplo más conocido son los *archivos*, que permiten utilizar indistintamente cualquier tipo de dispositivo de almacenamiento, mediante unidades lógicas manejables y asimilables para el usuario. Otros Recursos Virtuales conocidos son la memoria virtual, las paginas WWW y los sockets.
- **Control de Ejecución.**
El propósito del S.O. es ejecutar aplicaciones. Es necesario entonces, llevar un control de la ejecución y determinar los mecanismos para asegurar que los programas se ejecuten de forma correcta. La ejecución puede ser secuencial, paralela, compuesta, etc.
- **Administración de los Recursos Físicos.**
El manejo eficiente de la asignación de recursos, evitar o controlar conflictos y realizar tareas de mantención, es una parte muy importante de un S.O. Estas tareas suelen ser transparentes para el usuario, pero influyen directamente en la correcta ejecución de sus aplicaciones y en el desempeño de estas mismas.

- **Protección.**

En sistemas con múltiples usuarios o de uso específico, deben existir mecanismos de protección para evitar el uso inapropiado del sistema. Herramientas de seguridad y control deben estar disponibles tanto a nivel de sistema como a nivel de usuario, para asegurar el objetivo para el cual fue diseñado el sistema computacional.

- **Auditora.**

El registrar los eventos o transacciones que realiza el S.O. es un muy buen aporte para todas las funcionalidades antes vistas. El crear un canal de retroalimentación ayuda a corregir y auditar el uso que se le da al sistema.

2.3. Administración de CPU y Memoria

Como explicábamos en el apartado anterior, administrar los recursos físicos de un sistema computacional, es una tarea muy importante de los S.O. Los recursos más importantes del computador son la CPU y la memoria principal. Las técnicas de gestión de procesos y administración de memoria han evolucionado durante el tiempo, aumentando su complejidad y mejorando su desempeño. Es preciso aclarar ciertos conceptos que se consideran como **características** de los sistemas operativos.

- **Nivel de Multiprogramación.**

El concepto de Multiprogramación consiste en que varios programas se estén ejecutando al “mismo” tiempo. Esto significa que deben **compartir** el espacio de memoria principal disponible y el tiempo de procesamiento. Entonces, el Nivel de Multiprogramación será que tantos procesos pueda ejecutar simultáneamente. Estrictamente, multiprogramación se refiere solo a la capacidad de tener cargados varios programas en memoria principal ya que la ejecución es secuencial.

- **Sistemas por lotes.**

Sistemas que se **planifican** antes de ejecutar las tareas. Consisten generalmente en ordenar de forma conveniente los programas a ejecutar en una cola de espera para mejorar el tiempo de respuesta de algunas tareas.

- **Tiempo Compartido.**

División del tiempo de procesamiento en pequeñas unidades atómicas que pueden ser *usadas* por los procesos. La idea es que todos los procesos se ejecuten de forma “simultánea” para la concepción psicológica del usuario. El procesador realiza solo una instrucción de máquina a la vez, pero en tiempos imperceptibles para el hombre. Es posible entonces dar la ilusión de que los procesos se ejecutan paralelamente.

- **Multiprocesamiento.**

Esto es la capacidad de efectivamente ejecutar dos aplicaciones en el mismo tiempo. Esto, lógicamente necesita más de un procesador, y el nivel de multiprocesamiento va estar dado por el número de procesadores con que se cuente. Necesita de soporte especial de sincronización y de asignación de recursos.

- **Sistema Multiusuario.**

Tal como lo dice su nombre, estos sistemas tienen la cualidad de tener múltiples usuarios en interacción o no. Cada uno debe poder explotar las necesidades que requiere y para ello, necesitamos mecanismos especiales de protección.

- **Modo Dual**

Mecanismo de protección del sistema operativo. Se hace la distinción en dos modalidades de trabajo: el modo *supervisor* y modo *usuario*. El modo supervisor (modo kernel), tiene acceso directo a todos los recursos del sistema, mientras que el modo usuario debe enviar solicitudes al modo supervisor para utilizar estos recursos. De esta manera, se puede llevar un control eficiente de los recursos y se protegen los componentes del uso indebido o no planificado.

Linux en particular, tiene soporte para multiprogramación, multiusuario, multiprocesamiento, tiempo compartido, modo dual y sistema por lotes. MS DOS, por ejemplo carece de varias de estas cualidades.

2.3.1. Gestión de procesos

Un proceso es un programa (o parte de él) en ejecución. Esto implica que debe estar cargado en memoria principal, y que se le debe asignar tiempo de procesamiento parcial o completo. El S.O. debe ser capaz de crear y destruir procesos, como tarea básica. Los sistemas con soporte de multiprogramación deben poder suspender y reanudar procesos, y por ende hacer cambio de contexto eficiente del procesador. Además deben existir mecanismos de sincronización de procesos, de comunicación de procesos y si es necesario manejo de deadlocks. La sincronización de procesos se realiza mediante **planificadores**, que van a definir el estado del proceso.

2.3.1.1. Planificador de Largo Plazo

Controla el grado de multiprogramación. En pocas palabras, es quien carga el proceso en memoria. Consiste en una *cola de spool* que espera a que el procesador (y el Planificador de Corto Plazo) pueda atender a alguno de los procesos que contiene. Estos procesos no han sido iniciados, es decir, no han recibido tiempo de CPU aun.

2.3.1.2. Planificador de Corto Plazo

Es quien decide que proceso asignarle a la CPU. Consiste generalmente en una *cola listo* que indica que aquellos procesos están dispuestos a recibir tiempo de CPU. Se ejecuta muy seguido (cada vez que un proceso sale de la CPU) y es el responsable de la interactividad del sistema.

2.3.1.3. Planificador de Mediano Plazo

Cuando un proceso esta ocupando CPU, pero esta en espera (de que termine un proceso de Entrada/Salida por ejemplo), cae en el planificador de medio plazo. Consiste en una *cola de swapped-out* que controla el grado de multiprogramación aumentándolo o disminuyéndolo. Ocupa la técnica de swapping, para la información de los procesos. Cuando la *cola listo* se ve sobrepasada (overflow), los procesos con mayor demanda de memoria pasan al planificador de mediano plazo hasta que la *cola listo* se desocupe.

2.3.1.4. Estados de Procesos

Dependiendo en que planificador y en que fase se encuentren los procesos, estos tomarán distintos estados:

- Nuevo
El proceso aun esta en el planificador de largo plazo, esperando ser admitido por el procesador por primera vez
- Ejecutándose
El procesos se encuentra “dentro” de la CPU, y esta ejecutando instrucciones de máquina. Si se tiene un solo procesador, solo un proceso puede estar en este estado en un mismo tiempo.
- Listo
Se encuentra en el planificador de corto plazo, dispuesto a ser aceptado nuevamente por la CPU.
- Esperar
Se encuentra en el planificador de mediano plazo, esperando que algún evento ocurra, para ser gatillado nuevamente. Suelen ser operaciones de Entrada/Salida.

- Terminado
Ya utilizó todo el tiempo de CPU necesario para terminar su ejecución. El proceso será eliminado prontamente.

2.3.2. Gestión de Memoria Principal

Probablemente, la Gestión de Memoria principal es la parte más compleja de los S.O modernos. La principal idea de un *controlador de memoria*, es convertir direcciones lógicas utilizadas por el programa, a direcciones reales ocupadas por el dispositivo electrónico de memoria. Además el controlador debe asignar memoria a distintos procesos y a sus datos, ofrecer mecanismos de protección, utilizar técnicas de swapping, etc. En sistemas multiprogramados, se debe dividir la memoria en múltiples procesos, con gran tasa de variabilidad. Existen procesos con un tiempo de vida del orden del nanosegundo, mientras que otros duran días. Cada proceso tendrá entonces un espacio de memoria asignado para su carga y datos; este espacio, puede ser fijo o dinámico, con paginación, segmentación o ambos. Realmente, para este curso lo importante es tener claro que todo proceso tiene su propio espacio de memoria, y que si se solicita una dirección no válida para el contexto, se producirá un error (segmentation fault).

2.4. Filosofía de trabajo en Sistemas *NIX

Nos referimos como “Sistemas *NIX” a aquellos que mantienen la filosofía Unix como piedra angular de su desarrollo y uso. A continuación se dará una pequeña reseña histórica de Unix y su filosofía de trabajo.

2.4.1. Filosofía Unix

El sistema Linux es un fiel heredero de la tradición UNIX. UNIX fue creado por un par de programadores expertos (Ken Thompson y Dennis Ritchie) en AT&T como proyecto personal para aprovechar un pequeño computador en desuso. En consecuencia, el sistema resultó simple (lo de UNIX es un juego de palabras respecto de Multics, un complejo sistema operativo que estaba desarrollando General Electric; se refiere a que hay sólo un mecanismo para cada operación). Además, como sus creadores lo estaban desarrollando para su propio uso, el resultado más que un sistema de uso general es un excelente ambiente de desarrollo de software. UNIX fue un sistema multiusuario desde sus inicios, que incorporó una serie de innovaciones de Multics, como ser facilidades para crear nuevos procesos libremente y un sistema de archivos jerárquico. Una de las grandes innovaciones de UNIX consiste en considerar todo como archivos, entendidos éstos a su vez simplemente como secuencias de bytes sin mayor estructura.

Las primeras versiones de UNIX estaban escritas en lenguaje assembler, considerado en la época la única posibilidad de obtener rendimiento aceptable. Al corto andar, el sistema operativo se reescribió en un lenguaje de alto nivel diseñado de forma de dar flexibilidad similar al assembler al programador, y que por ser simple y cercano a la máquina resulta fácil generar código eficiente para él. Este es el origen del lenguaje C, en amplio uso (y también el lenguaje en que está escrito mayoritariamente el núcleo Linux).

En sus orígenes, UNIX sólo tenía interfaces de línea de comandos (los terminales gráficos de la época eran extremadamente caros). Como resultado de “todo es un archivo,” “es barato crear procesos,” y “la máquina (y el grupo de desarrollo) es limitada” en UNIX es tradicional crear *herramientas*, pequeños programas que se concentran en resolver un único problema. La idea es conectar estos programas especializados mediante *pipes* (tuberías). Como las entradas y salidas que éstos esperan y generan son simplemente secuencias de bytes, es simple arreglar estas conexiones. Así, combinando adecuadamente piezas ya existentes se pueden construir aplicaciones complejas. Sería raro contar con una operación rebuscada, como contar el número de archivos en el directorio actual que contengan la palabra “Unix,” sin embargo, crear ésta es muy simple:

```
find . -type f -print | xargs grep -l Unix | wc -l
```

Acá, `find(1)`³ ubica archivos (`-type f`) en el directorio actual, e imprime sus nombres (`-print`); la (posiblemente gigantesca) lista de nombres de archivo la toma `xargs(1)`, quien la troza en piezas manejables para entregarlas a `grep(1)` (búsqueda de un string en sus argumentos) con argumentos `-l` (sólo liste archivos que contienen el string) y `Unix`; la lista resultante la recibe `wc(1)` (cuenta palabras, líneas, y caracteres), quien en este caso (`-l`) sólo contará líneas. Para simplificar este tipo de usos los comandos en UNIX suelen ser parcos, no adornan sus salidas con encabezados ni resúmenes que posibles usuarios corriente abajo deberán remover trabajosamente.

Como resultado se obtiene cierta uniformidad de la interfaz: Cada vez que se desea paginar algún resultado, no es el programa que lo genera el encargado de paginar, sino un programa especializado para esta tarea (como es `more(1)`). Así, la forma de controlar la paginación será siempre la misma, además de simplificar el programa que genera los resultados en primer lugar. Uno de los efectos adicionales de esta forma de proceder es que proliferaron implementaciones distintas de las herramientas básicas, según las inclinaciones de sus autores (para desplegar archivos, están al menos `more(1)`, `less(1)`, y `view(1)`) y en algunos casos como resultados de avances de la tecnología (es el caso de la familia `grep(1)`).

Desde sus comienzos, AT&T usó UNIX extensamente en sus operaciones. Siendo un monopolio telefónico en la época, las leyes norteamericanas le vedaban el incursionar en otros negocios, como el desarrollo de software. En parte por ésto, y también como una forma de reclutar mano de obra para el desarrollo de UNIX, la licencia de UNIX era de muy bajo costo para universidades. Esto hizo que innumerables cursos de sistemas operativos se basaran en este sistema (pequeño y sencillo). Asimismo, se hizo popular como plataforma de trabajo e investigación, con lo que mucho software para UNIX se desarrolló y diseminó. Por ejemplo, buena parte del desarrollo de TCP/IP, base de Internet, se llevó a cabo en el ámbito de UNIX, que de esta forma continúa siendo la plataforma más natural para ofrecer servicios a través de la red.

2.4.2. GNU y Linux

El proyecto GNU apareció como una reacción a la tendencia creciente al comienzo de los años 1980 en las universidades norteamericanas de considerar software desarrollado en ellas como un producto, limitando la hasta entonces común práctica de compartirlo libremente. Uno de los grandes aportes de este proyecto es la licencia GPL (*General Public Licence*), que en lo principal indica que el software cubierto por ella puede usarse libremente para cualquier fin, y que en caso de distribuirse a terceros, éstos deberán recibir el código fuente completo del producto (de forma de permitirles a su vez modificarlo según sus necesidades), y no se les pueden imponer condiciones adicionales a la redistribución. La única condición para usar el software como base en la creación de productos derivados es que éstos deberán también distribuirse bajo las mismas condiciones. La idea es que software cubierto por GPL (y todo lo que de él resulte en virtud de esta licencia) esté siempre disponible para todos los interesados. Como ya se indicó, el proyecto GNU en sí tiene como objetivo crear un sistema compatible con UNIX libre en el sentido indicado. Este objetivo no se ha logrado hasta hoy, pero hay una impresionante cantidad de paquetes que implementan diversos aspectos de la funcionalidad de UNIX. Al ser los paquetes de GNU desarrollos independientes de funcionalidad ya existente, típicamente incorporan mejoras como mayor facilidad de uso o mayor generalidad. De especial interés son el compilador C, `gcc(1)` (herramienta indispensable en Linux, desarrollada durante muchos años como compilador para una gran variedad de arquitecturas usadas en sistemas empotrados por la empresa Cygnus), y el editor `emacs(1)` (más que únicamente un editor de texto es la pieza central de un riquísimo ambiente de trabajo, muy popular en UNIX). Muchos desarrolladores no asociados directamente al proyecto GNU han puesto sus creaciones bajo GPL, con la intención de asegurar que sus esfuerzos siempre estarán disponibles libremente. El mismo núcleo Linux, e incluso distribuciones completas como es el caso de Red Hat, se distribuyen bajo esta licencia.

³Usamos la convención de hacer referencia a los comandos y demás términos propios de UNIX dando su nombre y la sección del manual donde se describen (acá sección 1, comandos de uso general). Mayores detalles se darán más adelante, en la sección 3.

El particular modelo de desarrollo de Linux ha sido adoptado por variados proyectos independientes. La mística tras Linux y su popularidad han dado un fuerte impulso al desarrollo de software de distribución gratuita. El núcleo Linux ofrece una base estable sobre la cual montar los resultados del proyecto GNU y otro software de libre distribución, que incluye servidores maduros para la variedad de servicios que conforman Internet. De esta forma, hoy día Linux es una plataforma muy atractiva para servidores; por su bajo costo, buen rendimiento, y flexibilidad. Por otro lado, el software de libre distribución existente en su conjunto ofrece un excelente y flexible ambiente de desarrollo sobre Linux; y éste a su vez ofrece hoy día los fundamentos sobre los cuales construir ambientes de trabajo orientados a usuarios finales. Es así como en forma relativamente reciente han aparecido ambientes y aplicaciones gráficas para una variedad de tareas, incluyendo hoy día primeras versiones de ambientes de oficina integrados.

3. Shell y Herramientas

En esta máquina (un notebook con instalación más bien típica de Red Hat, aunque incluyendo herramientas de desarrollo), hay 2425 comandos de uso general, 513 comandos especializados de administración, y 171 comandos específicamente gráficos. Está claro que nadie puede manejarse libremente con tal cantidad de programas. Más aún, muchos de los programas tienen una gran cantidad de opciones. Esta sección resume los programas más importantes con sus opciones más usadas. El que algún comando esté incluido en esta breve presentación no significa que sea la única alternativa (o necesariamente la mejor) para una tarea particular. Para mayores detalles, y para referencia de los muchos comandos importantes que quedan fuera de la presente presentación use los manuales en línea.

Unas palabras de advertencia previa: UNIX parte de la base que el usuario *sabe* lo que está haciendo. Los comandos no solicitan confirmación antes de efectuar operaciones potencialmente dañinas. En mi experiencia tales solicitudes de confirmación pueden salvar el pellejo de un novato, pero un experto rápidamente automatiza el responder “Sí” a las consultas, con lo cual no se gana nada y sólo pierde tiempo. En un sistema multiusuario los recursos que un usuario (o proceso) libera pueden ser usados rápidamente por otros, con lo que “desborrar archivos” sólo puede tener éxito ocasionalmente. Por lo demás, acostumbrarse a que los archivos eliminados pueden restaurarse sólo lleva a que cuando el archivo sea *realmente* importante un usuario confiado lo borre, y justo en ese caso no es posible recuperarlo.

Los nombres de los comandos más usados en UNIX son crípticos, generalmente dos letras. Nuevamente, la justificación es que el sistema está orientado al experto que lo usa diariamente, con lo que los nombres no son importantes, y sí lo es la rapidez con la que pueden darse.

Muchos comandos aceptan opciones cortas (que se introducen con un guión, y típicamente constan de una sola letra) u opciones largas (introducidas por dos guiones). En lo que sigue se dan únicamente las opciones cortas, más fáciles de tipear. Hay casos en que alguna facilidad está disponible sólo en forma de una opción larga, pero éstas suelen ser bastante rebuscadas y escapan a esta presentación resumida.

Una convención común (pero no universal) es que dos guiones (--) indican el fin de las opciones y el comienzo de los argumentos, lo puede resultar importante al usar argumentos que comiencen con guión. Igualmente, muchos comandos dan una sinopsis de su uso al dárseles una opción desconocida, o en respuesta a la opción `--help` o `-h`.

3.1. Kit de supervivencia en la línea de comandos

3.1.1. El intérprete de comandos `bash(1)`

El programa que interpreta los comandos del usuario en UNIX no tiene nada de especial, es un programa corriente. Incluso se indica cuál es el elegido por cada usuario en la configuración de su cuenta, y el mismo usuario puede cambiarlo usando `chsh(1)`. Esto ha llevado a una proliferación de intérpretes de comandos, de los cuales acá sólo se describe brevemente el más usado, `bash(1)`.

El intérprete de comandos tradicional en UNIX es el llamado *Bourne shell*, desarrollado por Stephen Bourne. La funcionalidad que éste ofrece es la base de lo que especifica POSIX para el intérprete de comandos. Sin embargo, `sh(1)` es bastante espartano. Como parte del proyecto GNU se desarrolló un nuevo shell, agregando funcionalidades útiles de una variedad de shells diferentes. Así nace el *Bourne again shell*, juego de palabras entre el nombre del creador del shell original y *born again*, o sea, *renacido*. Entre las características destacables de `bash(1)` están el manejo de una historia de comandos; edición de la línea de comandos; completar nombres de comandos, archivos, y variables con `TAB`. Las facilidades de edición y búsqueda en la historia de comandos son afines a los comandos de `emacs(1)`, que se discuten bajo 3.3.1. Al ser `bash(1)` un programa muy complejo, acá sólo se discuten sus características más importantes.

3.1.2. Sistemas de ayuda

Dado un sistema con tal riqueza de recursos, es importante el sistema de ayuda. En Linux generalmente se encuentran varios:

- El sistema `man(1)`, tradicional de UNIX. Está basado en el sistema de procesamiento de texto `troff(1)`, con el cual se pueden generar por un lado manuales impresos de alta calidad y por el otro aproximaciones aptas para ser desplegadas en un terminal.
- El sistema `texinfo(5)`, del proyecto GNU. Está basado en el sistema de procesamiento de texto `TEX`, y permite generar tanto manuales impresos de alta calidad como colecciones de páginas enlazadas en hipertexto.

Estos dos sistemas son diferentes, el formato `man(1)` tradicional está orientado más a resúmenes de uso, mientras que `texinfo(5)` ofrece facilidades para crear verdaderos libros de texto sobre el tema a tratar. En todo caso, es posible obtener manuales tradicionales de forma semiautomática de la documentación en `texinfo(5)`, siempre que ésta esté organizada adecuadamente.

En lo que sigue se describen someramente los comandos principales para acceder a los manuales en línea de ambas opciones.

3.1.2.1. El sistema `man(1)`

Siguiendo la estructura de los manuales de UNIX, que se distribuían en ocho carpetas⁴ las actuales secciones del manual son:

Sección 1: Comandos de uso general.

Sección 2: Llamadas al sistema, vale decir, operaciones que el núcleo maneja directamente.

Sección 3: Rutinas de biblioteca, operaciones que se efectúan al menos parcialmente en el programa que las invoca.

Sección 4: Archivos especiales, más que nada descripción de dispositivos y su manejo.

Sección 5: Formatos de archivo y diversos protocolos.

Sección 6: Juegos y entretenimientos.

Sección 7: Convenciones y miscelánea.

Sección 8: Comandos de administración del sistema.

Sección 9: Aspectos internos del núcleo.

Sección l: Páginas de agregados locales.

Sección n: Páginas nuevas.

Sección o: Páginas antiguas (*old*).

Las secciones 8 y 9 son nuevas. En la sección 8 se separaron las páginas de manual de los comandos de administración de los comandos de uso general por un lado para evitar confundir a los usuarios normales y por el otro para disminuir el volumen de la sección 1. La sección 9 es aún más reciente, tradicionalmente las interfaces internas del núcleo no estaban documentadas (en un UNIX comercial sólo están disponibles interfaces para agregar nuevos dispositivos). Las secciones identificadas por letras no son estándar, aunque existen en muchas instalaciones.

⁴Un dicho clásico es que “UNIX trae suficientes manuales para matar una vaca”

Cabe hacer notar que ciertos UNIX se apartan de este esquema, organizando los manuales bajo letras (es el caso de SCO) o agregando subdivisiones marcadas por letras, por ejemplo en Solaris está la sección `1x`, manuales de comandos de uso general (1) gráficos (usan el sistema X Windows).

Las páginas se restringen a detallar cómo exactamente se usan las facilidades que describen. No son tutoriales, se supone que esa función la cumplen otras instancias. Tienen la estructura general siguiente, cuyas partes no siempre están presentes:

Encabezado: Da el nombre y la sección de la página.

Nombre: Da un breve resumen del tópico tratado.

Sinopsis: La intención es servir de recordatorio a quien sólo desea consultar un detalle. Especifica cómo se usa el comando (resumen de opciones y argumentos) o la rutina (prototipo y forma de llamarla desde C).

Descripción: Da una descripción más detallada del tema.

Opciones: Describe en detalle cada una de las opciones que reconoce el comando.

Archivos: Archivos que usa el comando, de ser aplicable.

Bugs: Posibles problemas conocidos, incompatibilidades con los estándares reconocidos, o infelicidades del diseño se documentan acá.

Vea también: Cada página se refiere exclusivamente a su tema particular, para enlazar páginas relacionadas se usa esta sección. En muchos casos además de otras páginas de manual hace referencia a material externo.

Para leer los manuales hay varias opciones. Los ambientes gráficos Gnome y KDE incorporan sistemas de ayuda gráficos propios, que a su vez permiten también leer los manuales. Parte del sistema gráfico X Windows es `xman(1)`, que ofrece una interfaz gráfica simple para elegir y desplegar los manuales, fiel a su objetivo de ser todo para todos el editor `emacs(1)` incluye facilidades para leer los manuales, etc.

En el caso de comandos se usa la convención general de indicar opciones entre corchetes, e indicar posibles repeticiones mediante elipsis (...). Se indica mediante subrayado el sentido general de parte de la descripción. El comando tradicional para leer los manuales en línea es `man(1)` (que despliega la página solicitada, opcionalmente de la sección dada). El formato más común es:

```
man [-a] [sección] nombre...
```

O sea, el nombre del comando es `man`, toma la opción `-a` (muestre todas las páginas de ese nombre, no sólo la primera), opcionalmente la sección y al menos el nombre de una página a desplegar. Un comando íntimamente relacionado es:

```
apropos clave...
```

Este muestra los resúmenes de las páginas de manual que incluyen alguna de las claves indicadas. Es útil para ubicar manuales respecto de algún tópico particular.

Como ya se indicó antes, en UNIX se ha evitado duplicar funciones. Para mostrar las páginas, que rara vez caben en una pantalla, `man(1)` usa `more(1)` para desplegar el texto:

```
more [+num] [archivo...]
```

La idea es desplegar a partir de la línea `num`, los archivos indicados. Si no se indican archivos, `more(1)` despliega su entrada página a página. Es posible avanzar pantalla a pantalla usando la barra espaciadora, una línea a la vez usando `<RETURN>`, y avanzar buscando un string mediante `/string`. En todo caso,

`more(1)` es bastante primitivo, el equivalente GNU, `less(1)` agrega funcionalidad como poder avanzar y retroceder en el archivo desplegado.

Una alternativa adicional la ofrece `emacs(1)`, a través de su comando `man` (invocado vía `M-x man`). Este incluso ofrece buscar la página para la palabra cercana al cursor, lo que es muy cómodo al programar.

3.1.2.2. El sistema `texinfo(5)`

Como parte del proyecto GNU se definió un sistema de documentación basado en hipertexto, capaz de acomodar no sólo descripciones concisas de los comandos, sino incluso libros de texto completos sobre el particular. Uno de los objetivos es hacer autocontenidos los manuales, no tener que recurrir a fuentes de información externas.

La organización de los manuales en este sistema es en forma de un menú que ofrece varias secciones organizadas por función. Cada ítem del menú está descrita por una entrada de la forma siguiente:

```
* nombre:: Descripción.
```

Esta forma hace referencia a una entrada de menú en el archivo presente. Para crear un menú enlazando entradas en diversos archivos se usa la forma:

```
* nombre: (archivo). Descripción breve del ítem.
```

Acá el `nombre` identifica al ítem, y el `archivo` indica el archivo que describe el ítem nombrado. En algunos casos la entrada toma la forma:

```
* nombre: (archivo)ítem. Descripción breve.
```

con lo que se hace referencia a un `ítem` al interior del archivo mencionado. Cada archivo `info` está organizado como menús que llevan a las distintas opciones, incluyendo referencias cruzadas.

Para navegar a través de estos manuales hay nuevamente varias opciones, que también incluyen a los sistemas de ayuda propios de los ambientes gráficos. El ambiente nativo para acceder a la ayuda es el ambiente `emacs(1)`, pero también hay comandos especiales:

```
info [ítem]
```

El comando `info(1)` muestra el árbol completo, o desde el ítem solicitado. Para navegar se usan los comandos `m` para entrar en un ítem específico, flechas para moverse a través del texto, `n` para avanzar al ítem siguiente, `p` para ir al ítem previo, `u` para subir un nivel, `f` para seguir una referencia cruzada (marcada mediante `(*note: ítem)`). El comando `g` permite ir a un nodo cualquiera dentro de la red, y `?` ofrece una breve ayuda (`info(1)` tiene muchos comandos adicionales, este resumen sólo indica los más usados).

Una alternativa algo más cómoda la ofrece `pinfo(1)`, que muestra las opciones resaltadas mediante colores y permite seleccionar vía mouse, además de los comandos indicados anteriormente. En caso de no hallar el nodo solicitado, `pinfo(1)` despliega la página del manual correspondiente.

3.1.3. Comandos Básicos

Los comandos más importantes para manejarse dentro del Sistema son los que se describen a continuación.

3.1.3.1. `ls(1)`

```
ls [-alRd] [archivo...]
```

Este comando muestra información sobre archivos, con `-d` en vez de mostrar información sobre los archivos contenidos en un directorio muestra información sobre el directorio. Las opciones `-aLR` indican mostrar información sobre archivos “ocultos” (cuyos nombres comienzan con un punto), dar información larga (incluyendo dueño, grupo, tamaño, y permisos), y mostrar recursivamente los archivos dentro de los directorios.

3.1.3.2. `rm(1)`

```
rm [-ifr] archivo...
```

Elimina archivos. Con `-i` solicita confirmación, con `-f` simplemente efectúa las operaciones indicadas sin chistar. La opción `-r` indica eliminar el directorio y todo su contenido.

3.1.3.3. `cp(1)`

```
cp [-ifr] fuente... destino
```

Copia archivos. Si hay varias fuentes, el destino debe ser un directorio. Con `-i` solicita confirmación, con `-f` simplemente efectúa las operaciones indicadas sin chistar. La opción `-r` indica copiar recursivamente el directorio y todo su contenido.

3.1.3.4. `mv(1)`

```
mv [-if] fuente... destino
```

Mueve archivos. Si hay varias fuentes, el destino debe ser un directorio. Con `-i` solicita confirmación, con `-f` simplemente efectúa las operaciones indicadas sin chistar. Si fuente y destino están en el mismo directorio, el efecto es simplemente cambiar el nombre.

3.1.3.5. `ln(1)`

```
ln [-ifs] fuente... destino
```

Enlaza archivos, vale decir, les agrega un nuevo nombre bajo el destino para las fuentes. Si hay varias fuentes, el destino debe ser un directorio. Con `-i` solicita confirmación, con `-f` simplemente efectúa las operaciones indicadas sin chistar. La opción `-s` especifica crear enlaces simbólicos, no enlaces duros. Nótese que no está permitido crear enlaces duros a directorios.

3.1.3.6. `mkdir(1)`

```
mkdir [-p] directorio...
```

Crea nuevos directorios. Con `-p`, crea también los directorios intermedios que sean necesarios.

3.1.3.7. `chown(1)`

```
chown dueño[:grupo] archivo...
```

Cambia dueño (y opcionalmente grupo) de los archivos indicados. Bajo Linux sólo `root` tiene permiso de cambiar el dueño.

3.1.3.8. chgrp(1)

```
chgrp grupo archivo...
```

Cambia grupo de los archivos indicados.

3.1.3.9. chmod(1)

```
chmod [+]= [rwxst] [ugoa] archivo...
```

Cambia permisos indicados de los archivos. Con += se indica agregar, quitar o dejar exactamente como se indican los permisos. Los permisos `rwxst` son lectura, escritura, ejecución, ejecución como el dueño o un miembro del grupo, y sólo permiso para el dueño de modificar archivos en un directorio. Las letras `ugoa` se refieren al dueño, al grupo, a otros, y a todos. Nótese que estas se pueden repetir, incluso varias operaciones separadas por comas. Así, para permitir lectura y escritura para el dueño y el grupo, negar lectura a otros, y permitir ejecución a todos del archivo `pgm` se puede usar:

```
chmod +rwug,-ro,+xa pgm
```

3.1.3.10. tar(1)

```
tar cmds [archivo...]
```

El comando `tar(1)` sirve para archivar, fue pensado para usarse con cintas. Sin embargo, se usa frecuentemente para empaquetar archivos. La sintaxis es un tanto extraña, los comandos a dar no son opciones. En los comandos debe ir exactamente uno de `c` (crear un nuevo archivo conteniendo los archivos y directorios indicados), `x` (extraer los archivos indicados, si no se dan archivos a extraer extrae todo), y `t` (dar la tabla de contenido del archivo). Pueden darse los modificadores opcionales `z` (use `gzip(1)` para comprimir o descomprimir), `j` (use `bzip2(1)` para comprimir o descomprimir), `v` (opere en forma verbosa). Además puede darse `f` para indicar un archivo sobre el cual trabajar, que en tal caso se especifica antes de los archivos. Si el nombre de este archivo es `-`, toma la entrada estándar o escribe en la entrada estándar.

3.1.3.11. wc(1)

```
wc [-cwl] [archivo...]
```

Cuenta caracteres (`c`), palabras (`w`), o líneas (`l`) en los archivos dados, o en su entrada.

3.1.3.12. cat(1)

```
cat [-n] [archivo...]
```

Concatena los archivos dados. Puede usarse `-` como nombre de archivo para incluir la entrada en alguna posición. Con `-n` numera las líneas.

3.1.3.13. sort(1)

```
sort [-rnu] [-t SEP] [-k POS1[,POS2]...] [-o salida] [archivo...]
```

Ordena los archivos indicados línea a línea. La opción `-r` indica ordenar de mayor a menor, `-n` especifica orden numérico, `-u` solicita que se retenga únicamente una de cada grupo de líneas iguales. La opción `-t` indica el caracter a usar para delimitar campos (por omisión es la transición entre no espacios y espacios). Con `-k` se indica el rango a ser usado para comparar, con `POS` una posición de la forma `F[.C][Q]`, donde `F` es un número de campo y `C` es un caracter dentro del campo (ambos contando desde uno), y `Q` es una opción de ordenamiento, que tiene precedencia sobre la opción global para ese rango.

3.1.3.14. `tr(1)`

```
tr [-cdst] CONJ1 [CONJ2]
```

Traduce, comprime, o elimina caracteres de la entrada estándar para dar la salida estándar. Con `-c` primero complementa el `CONJ1`, con `-d` simplemente elimina caracteres del `CONJ1`, con `-s` condensa secuencias de caracteres en uno solo, con `-t` primero trunca el `CONJ1` al tamaño del `CONJ2` (en caso contrario, caracteres sin traducción simplemente se descartan). Los conjuntos de caracteres se dan como strings, pudiendo indicarse rangos como `CHAR1-CHAR2`. Por ejemplo, lo siguiente traduce mayúsculas en minúsculas:

```
tr 'A-Z' 'a-z'
```

3.1.3.15. La familia `grep(1)`

```
grep [-vcqFGE] [-B NUM] [-A NUM] patrón [archivo...]
```

Muestra las líneas de los archivos que calzan con el `patrón`. Con `-B` muestre este número de líneas antes del calce, con `-A` este número de líneas después. Con `-v` se especifica mostrar las líneas que *no* calzan, `-q` indica sólo indicar si se encontró o no el patrón, `-l` solicita mostrar únicamente los nombres de los archivos en que hay calce. Las opciones `-FGE` especifican patrones fijos (comportamiento del programa `fgrep(1)` tradicional), patrones tradicionales (como del programa `grep(1)`), y patrones extendidos (como de `egrep(1)`).

3.1.3.16. `cmp(1)`

```
cmp [-s] archivo1 archivo2
```

Compara los archivos, muestra dónde difieren por primera vez. Con `-s` sólo retorna éxito (archivos iguales) o falla (diferentes).

3.1.3.17. `diff(1)`

```
diff [-ucr] archivo-de archivo-a
```

Compara dos archivos (o con `-r`, recursivamente los archivos contenidos en dos directorios) dando las diferencias. Con `-c` da contexto a las diferencias, con `-u` entrega diferencias unificadas.

3.1.3.18. `patch(1)`

```
patch [-pnum] [-NR] [original [parche]]
```

Aplica el resultado (parche) de `diff(1)` al archivo original. Las opciones `-NR` indican orden normal (hacia adelante) y `reverso` (hacia atrás, el parche se creó dando original y nuevo invertidos o se quiere deshacer un parche aplicado antes). De no darse parche, éste se toma de la entrada estándar, si no se da original los nombres de los archivos originales se determinan del parche mismo. En tal caso, `-pnum` indica que deben eliminarse num directorios del comienzo de los nombres de archivo antes de aplicar los parches. Si vienen varios parches en la entrada, `patch(1)` intenta aplicarlos en secuencia, y omite basura antes y después de los parches mismos (de esta forma, es posible aplicar directamente un mensaje de correo con encabezados, una explicación, y el parche mismo).

3.1.3.19. `gzip(1)`

```
gzip [-cdlt] [-#] [archivo...]
```

Este comando comprime o descomprime (`-d`) archivos. Al comprimir deja `archivo.gz` por omisión, borrando el original. Es capaz de descomprimir archivos generados con `compress(1)` (extensión `.Z`), aunque no de generar tales archivos (en todo caso, `compress(1)` comprime mucho menos que `gzip(1)`). Con `-#` se indica un nivel de compresión, de 1 a 9. Con `-c` escribe en la salida estándar, no crea el archivo comprimido. Si no se dan archivos, opera sobre la entrada estándar y da el resultado en la salida estándar. Con `-l` lista el contenido de un archivo comprimido, dando estadísticas de compresión, con `-t` verifica la integridad del archivo comprimido. Invocado como `zcat` descomprime a la salida estándar, como `zmore(1)` descomprime y pasa el resultado a `more(1)`.

3.1.3.20. `bzip2(1)`

```
bzip2 [-cdlt] [-#] [archivo...]
```

Este comando comprime o descomprime (`-d`) archivos. Comprime mejor que `gzip(1)`, pero requiere más recursos. Al comprimir deja `archivo.bz2` por omisión, borrando el original. Con `-#` se indica un nivel de compresión, de 1 a 9. Con `-c` escribe en la salida estándar, no crea el archivo comprimido. Si no se dan archivos, opera sobre la entrada estándar y da el resultado en la salida estándar. Con `-l` lista el contenido de un archivo comprimido, dando estadísticas de compresión, con `-t` verifica la integridad del archivo comprimido. Invocado como `bzcat(1)` descomprime a la salida estándar.

3.1.3.21. `file(1)`

```
file archivo...
```

Intenta determinar el tipo del contenido de cada uno de los archivos indicados.

3.1.3.22. `find(1)`

```
find directorio...expresión
```

Busca archivos que cumplan la expresión en los directorios indicados. Para detalles de cómo componer la expresión véase el manual. Básicamente es una secuencia de condiciones que se verifican una a una, uniéndolas con `y` lógico por omisión. Casos simples son `-type [fd]` (es un archivo o directorio, respectivamente), `-newer archivo` (es más nuevo que archivo), `-empty` (el archivo o directorio está vacío), `-print` es siempre verdadero e imprime el nombre del archivo. Con `-not` se niega la expresión siguiente.

3.2. Conceptos Avanzados

3.2.1. Comandos y pipelines

En UNIX todo programa inicia su ejecución conectado a tres archivos: Su entrada estándar (normalmente la entrada desde el teclado), su salida estándar (normalmente desplegada en pantalla), y la salida de errores estándar (normalmente también conectada a la pantalla, pero independiente de la anterior). Una de las tareas que lleva a cabo el shell es conectar programas entre sí.

Las opciones más simples corresponden a redireccionar un archivo a la entrada del programa:

```
pgm < entrada
```

con lo que éste toma la entrada del archivo y no del teclado. Similarmente, se puede redireccionar la salida a un archivo:

```
pgm > salida
```

con lo que la salida aparece en el archivo (posibles mensajes de error se dirigen a la salida de error estándar, asociada a la pantalla, no aparecerán en el archivo). Una variante agrega la salida al archivo, no lo sobrescribe:

```
pgm >> salida
```

Está también la posibilidad de recoger salida y errores por separado:

```
pgm > salida 2> errores
```

o en conjunto:

```
pgm &> todo
```

Además, está la posibilidad de conectar la salida de un programa como la entrada de otro:

```
pgm1 | pgm2
```

Estas operaciones se pueden encadenar y combinar a gusto, lo que da un mecanismo muy poderoso para construir operaciones complejas a partir de las operaciones simples que ofrecen las herramientas propias de UNIX. A una colección de procesos encadenados de esta forma se le llama *pipeline*.

Otras de las opciones que ofrece `bash(1)` corresponden al manejo de procesos. Por omisión, `bash(1)` espera a que el pipeline complete su proceso antes de solicitar nuevos comandos del usuario. Esto se puede evitar enviando los procesos a *background*, indicado con `&`:

```
pgm1 < datitos | pgm2 > resultados &
```

Estos procesos seguirán corriendo, incluso después que el usuario se desconecte del sistema. Una opción adicional que ofrece `bash(1)` es lo que se conoce como *job control*, la posibilidad de manipular los comandos. Con `ctrl-Z` se detiene el pipeline actual, pudiendo enviarlo al background con el comando `bg`, traerlo de vuelta al foreground con `fg`. Es posible tener varios pipelines suspendidos, y elegir entre ellos por número. El comando `jobs(1)` muestra las tareas en ejecución.

3.2.2. Interpretación de líneas

Inicialmente el shell separa las líneas en palabras delimitadas por espacios o TAB. A las palabras resultantes se les aplican sucesivamente varias expansiones, en el orden indicado más abajo, el resultado pasa como una lista de palabras al programa, quien es libre de interpretarlas a su antojo.

En primera instancia se expanden construcciones de la forma `{alt,er,nativa,}`, vale decir, alternativas separadas por comas entre llaves. Así, `{a,b,c}x{1,2}` resulta en `ax1`, `ax2`, `bx1`, `bx2`, `cx1`, y `cx2`. Nótese que no es necesario que existan archivos con los nombres resultantes.

Enseguida, se expanden construcciones que contienen tildes, en particular `~` se expande al nombre del directorio del usuario, y `~usuario` se expande al nombre del directorio de `usuario`.

El shell maneja variables, de nombres alfanuméricos. Para asignar un valor a una variable se usa la construcción

```
var=valor
```

donde `var` es el nombre de la variable, y `valor` es el valor que se asigna. Para hacer referencia al valor de la variable se usa `$var`. Para distinguir el nombre de la variable de caracteres alfanuméricos que pudieran estar a continuación, se escribe por ejemplo `${var}iable`. Variantes útiles son `${var#string}`, que da el valor de `$var` sin `string` al comienzo, `${var%string}`, que elimina `string` al final. Otra construcción útil es `${var:-string}`, que toma el valor de `$var` si lo tiene y no es nulo, o `string` en caso contrario. Se pueden tener arreglos, cuyos elementos se referencian vía `${arreglo[indice]}`. Acá `índice` es una expresión numérica de valor cero o mayor. Se puede asignar al arreglo completo vía

```
arreglo=(elem1, elem2, ...)
```

donde cada elemento tiene la forma `[índice]=valor`, siendo obligatorio sólo el `valor`.

Tienen significado especial los caracteres `*` (significa “cualquier cosa” en el ámbito de nombres de archivo), `?` (un caracter cualquiera en nombre de archivo), `[]` (lo que hay entre corchetes es un conjunto de caracteres). Lo que hace el shell con ellos (la operación de *globbing*) depende de los nombres de los archivos que estén presentes. Por ejemplo, `a*b` se expandirá a los nombres de todos los archivos cuyos nombres comiencen con `a` y terminen con `b`, sin importar lo que haya entremedio. De la misma forma, `A???` se expandirá a los nombres de archivos que comiencen con `A`, y que tengan largo exactamente cuatro; mientras `x[1-3]` se expande a los nombres de aquellos archivos que se llamen `x1`, `x2`, o `x3`. Estas construcciones pueden combinarse, o sea, `?f*` se expande a los nombres de todos los archivos cuyo nombre tiene `f` como segundo caracter. En caso de no haber archivos cuyos nombres calcen con el patrón, simplemente queda el patrón tal cual.

Otras substitutiones son las aritméticas, en las cuales se calculan los valores de expresiones como `[$i + 17] * $j` y las substitutiones de comandos, que se expanden a una palabra por línea de salida del comando en `$(comando)` o `'comando'`. La primera forma es preferible, dado que puede anidarse. La segunda es estándar de `sh(1)`.

Finalmente, resulta necesario citar texto que contiene caracteres con significado especial. Lo básico es simplemente preceder el caracter del caso con backslash. Así, `\$var` es simplemente eso, no el valor de la variable. Similarmente, en texto cualquiera entre comillas sólo se substituyen variables, y se efectúan expansiones aritméticas y de comando; mientras texto entre apóstrofes queda tal cual.

El manejo de las diversas expansiones es bastante más complejo de lo indicado acá, hay una variedad de alternativas adicionales además de situaciones como usar patrones y substitutiones en los nombres de las variables. En todo caso, esto cubre largamente lo usado frecuentemente.

3.2.3. Estructuras de control

El intérprete de comandos ofrece todo un lenguaje de programación. Primeramente, las instrucciones individuales se separan mediante punto y coma(;) o pasando a una nueva línea.

La base para la lógica es que un pipeline que falla se considera falso, si tiene éxito verdadero. Los resultados se pueden combinar usando `&&` (y lógico) y `||` (o lógico). Además, las expresiones así construidas se evalúan sólo hasta conocer el resultado (éxito o fracaso). De esta forma es común ver construcciones como:

```
condición && acción
```

donde se evalúa la *condición*, y sólo si es verdadera (tiene éxito) se efectúa la *acción*. Para condiciones es cómodo el comando `test`, también escrito `[...]`. Pueden usarse las siguientes condiciones, entre muchas otras:

-f archivo El archivo nombrado existe.

-x archivo El archivo nombrado existe y es ejecutable.

-z string El string tiene largo cero.

-n string El string tiene largo no cero.

string1 == string2 Los strings son iguales.

string1 != string2 Los strings son diferentes.

Así suelen verse idiomias como:

```
[ -f /etc/configuracion ] && comando
```

para ejecutar el *comando* sólo si su archivo de configuración existe. Más tradicionales son las estructuras `if`, `while`, y `until`:

```
if condición1; then
    acción1
elif condición2; then
    acción2
else
    acción3
fi
```

```
while condición; do
    acción
done
```

```
until condición; do
    acción
done
```

Para iterar sobre una lista de valores está:

```
for var in val1 val2 ...; do
    acción
done
```

La selección múltiple está dada por:

```
case string in
    patrón) acción;;
    patrón|patrón) acción; acción;;
    *) acción;;
esac
```

Los *patrones* se intentan calzar contra el `string` uno a uno, las acciones del primero que tenga éxito se ejecutan (nótese que cada caso termina con doble punto y coma). Se usa `*` (que siempre calzará) para proveer una acción por defecto.

Finalmente está la posibilidad de definir funciones, que se llaman como programas comunes:

```
function f() { ... }
```

En el cuerpo de la función se puede hacer referencia a variables, que se consideran globales. Para referirse a los argumentos de una función se usan las variables `$1`, `$2`, etc. También se pueden manejar usando la operación `shift`, que desplaza los argumentos una posición hacia la izquierda y tiene éxito si quedaban argumentos.

3.2.4. Entrada y salida

Se puede leer una línea en una variable mediante

```
read var
```

Esta operación tiene éxito si se logró leer una línea, falla cuando se alcanza fin de archivo. Para salida puede usarse la operación `echo`, que simplemente escribe sus argumentos, con la salvedad que `echo -n` escribe sin pasar luego a la línea siguiente de la salida (útil para solicitar ingreso de datos).

3.2.5. Scripts

Los sistemas UNIX modernos reconocen archivos ejecutables que comienzan con los caracteres `#!` (*shebang*) como *scripts*, archivos de comandos a ser ejecutados por el programa indicado en esa primera línea. Un script elemental sería:

```
#!/bin/bash
# Este es un ejemplo muy simple de script
echo Hay $# argumentos

for a; do
    echo $a
done
```

Puede usarse `#` para iniciar comentarios, la variable `$#` es el número de argumentos, y se muestra la variante de `for` sin `in`, que itera sobre los argumentos.

3.2.6. Configuración de la cuenta

Al iniciarse, `bash(1)` ejecuta el archivo `.bashrc` en el directorio de la cuenta, y si este proceso resulta directamente de conectarse al sistema también `.bash_profile`. En estos archivos tradicionalmente se definen variables de ambiente que controlan el funcionamiento del sistema. También es común definir *alias*, abreviaturas para comandos.

3.3. Editores

3.3.1. El editor `emacs(1)`

Uno de los productos principales del proyecto GNU es el editor `emacs(1)`. Más que un simple editor de texto, es el centro de todo un ambiente de trabajo, con modos especiales para editar archivos en una amplia gama de lenguajes, posibilidades de controlar depuradores desde el mismo editor, leer correo, e incluso navegar por Internet. Según sus fanáticos, la única deficiencia que tiene es que hace un café horrible...

Si se ejecuta `emacs(1)` en ambiente gráfico, aparece una barra de tareas en el borde superior. En el botón marcado `Help` aparece un menú que incluye acceso a un tutorial. Este tutorial, y los manuales en línea del editor son una buena forma de aprender a usarlo.

Algunas convenciones con las que es indispensable familiarizarse son `C-x` para referirse a `ctrl-X`, `M-x` para referirse a la combinación `alt-X` (o `meta-X` en algunos teclados, cuyo efecto también se obtiene vía `ESC X`). Algunas teclas tienen nombres especiales, como `ESC`, `SPC` (espacio) y `TAB` (tabulador).

En general, `emacs(1)` inserta los caracteres escritos en el archivo, para operar se usan caracteres de control y otras combinaciones. Algunos comandos importantes son `C-x C-f` (`find-file`, abre un nuevo archivo), `C-x C-s` (`save-buffer`, guarda el archivo que se está editando actualmente), `C-x C-w` (`write-file`, escribe el archivo bajo el nombre indicado), `C-_` (`undo`, deshace la última modificación), y `C-x C-c` (`save-buffers-kill-emacs`, ofrece guardar cada uno de los archivos que se están editando actualmente y luego termina). Debe tenerse cuidado con `C-z`, esto sólo suspende el editor, no termina el proceso.

Para moverse por el archivo se pueden usar las flechas, `PgDn` y `PgUp`, y la barra de desplazamiento. Con `M-<` se va al comienzo del archivo, con `M->` al final. Para búsquedas hacia adelante se usa `C-s`, hacia atrás `C-r`, y para reemplazos `M-%`. Con `C-d` se borra el carácter bajo el cursor, `C-k` borra hasta el final de la línea.

Con `C-SPC` se sitúa una marca en la posición actual, varios comandos trabajan sobre la *región*, el rango entre la posición actual y la marca. Por ejemplo, `C-w` borra la región.

Acceso al sistema de ayuda se obtiene mediante `C-h`, con una variedad de opciones adicionales. Por ejemplo, `C-h t` lleva al tutorial ya mencionado, `C-h i` lleva a leer los manuales en línea, `C-h k` muestra documentación para una secuencia de teclas.

3.3.2. El Editor `vi(1)`

`Vi` es un editor de texto interactivo, cuya peculiaridad es que funciona bajo dos modos: uno para ingresar texto y otro para ingresar comandos; el usuario debe alternar entre ambos modos según lo que desee realizar. Este detalle hace que `vi` sea algo difícil de aprender para los usuarios novatos, pero a la vez lo hace un editor bastante versátil debido a la gran cantidad de comandos que ofrece.

`vi` es el único editor que viene por defecto en todas las distribuciones de Unix, por lo que es importante conocer sus comandos básicos. Las distribuciones más recientes de Linux traen programas clones de `vi` (como `vim`, `elvis`, `vile`, etc.) que incorporan muchas más funcionalidades, haciendo de ellos editores extremadamente poderosos, sin que ello signifique consumir muchos recursos, como lo hacen editores

como emacs⁵.

Al entrar a vi, se ingresa en el modo de comandos. Para entrar al modo de comandos desde el modo de edición, basta presionar Esc. A continuación, veremos algunos comandos básicos:

- Comandos tipo ed (terminados con un salto de línea)

```
:w archivo guarda el archivo actual con el nombre señalado
:w          guarda el archivo actual
:q         sale de vi (no recomendado)
:q!        sale de vi perdiendo todos los cambios realizados
:e archivo abre el archivo señalado
:new archivo abre el archivo señalado en una nueva ventana
:help comando da ayuda sobre un comando específico (sólo en vim)
:!comando ejecuta un comando externo
:r!comando ejecuta un comando externo y pega la salida en vi
```

- Comandos de movimiento

```
h j k l mueve el cursor hacia la izquierda, hacia abajo, hacia arriba y hacia la derecha, respectivamente
w e     avanza al principio y al final de la palabra, respectivamente
b       mueve el cursor a la palabra anterior
Control-F avanza una página hacia adelante
Control-B avanza una página hacia atrás
0 $     se mueve al principio y al final de la línea, respectivamente
{ }     se mueve al principio y al final del párrafo, respectivamente
```

- Comandos para entrar al modo de edición

```
i entra al modo de edición con el cursor en la posición actual
a entra al modo de edición con el cursor en la posición siguiente
A entra al modo de edición con el cursor al final de la línea
```

- Comandos de edición

```
dd cortar la línea actual
d3d cortar 3 líneas
diw corta la palabra actual
yy copia la línea actual
cw cambia la palabra actual por el texto a ingresar
p pega en la posición siguiente
P pega en la posición actual
x elimina un caracter
u deshacer
Control-R
. repetir
```

- Comandos de búsqueda (terminados con un salto de línea)

```
/patrón busca el patrón hacia adelante
?patrón busca el patrón hacia atrás
:%s/abc/def/g reemplaza el patrón "abc" por el patrón "def" en todo el archivo
n busca la siguiente ocurrencia del último patrón buscado
* busca la siguiente ocurrencia de la palabra bajo el cursor
```

⁵Se suele decir que vi es un editor texto, mientras que emacs es un "entorno de escritorio que trae un editor de texto"

3.3.3. Otros

Existe una gran cantidad de editores especializados para distintas tareas como desarrollo de aplicaciones, redactar correos electrónicos, etc.. Dentro de estos encontramos por ejemplo gedit, kedit y pico ⁶. Existen también editores minimales para tareas de administración como lo es nano. Por otro lado existen grandes entornos de desarrollo que también pueden ser clasificados como editores, como por ejemplo Kdevelop o Anjuta.

⁶Pine Composer

4. GNU/Linux por dentro

4.1. Bootloaders

Lo primero que realiza el computador al encenderse es un test para comprobar que todas sus partes estén trabajando bien, memoria, cpu, discos, etc. Después del test, un programa llamado bootstrap loader, localizado en la memoria de la BIOS comienza a buscar un sector de booteo correspondiente al primer sector de un disco o diskette, actualmente también se busca en CDRoms y a través de la red. Este sector contiene un pequeño programa que puede cargar el sistema operativo.

Cuando encuentra un sector de booteo, lo lee y carga en memoria, pasándole el control para que se encargue del sistema operativo. En nuestro caso LILO o Grub first stage loader, pequeño programa cuya única función es llamar al second stage boot loader. Este segundo programa nos entrega un prompt que nos permite seleccionar el sistema operativo a cargar, en el caso de que se tenga más de uno o varias versiones de kernel instaladas.

Nota: Cuando el sistema está funcionando y se ejecutan los programas lilo o grub, lo que se está haciendo es ejecutar un map installer que lee la configuración de booteo desde un archivo y la escribe en el primer sector del disco.

4.1.1. Configuración de Bootloader: grub y lilo

El archivo de configuración de lilo es /etc/lilo.conf y existe una página de manual que describe sus detalles, tiene un aspecto como el siguiente:

```
prompt
timeout=50
default=linux
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
message=/boot/message
linear

image=/boot/vmlinuz-2.4.18-3
label=linux
read-only
root=/dev/hda1

label=otro.s.o
read-only
root=/dev/hda3
```

y entrega información sobre el kernel, la partición raíz, entre otras cosas, y puede manejar la partida de varios sistemas operativos, en el caso anterior, otro.s.o hospedado en la partición /dev/hda3. Se carga ejecutando: >lilo

Por su parte, grub guarda su configuración en /etc/grub.conf, archivo que contiene información como la que sigue:

```
default=0
timeout=10
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
password --md5 $1$k1jkdgh.\$4%45kj765fHGFBzd2
```

```
title Red Hat Linux (2.4.18-3)
    root (hd0,0)
    kernel /boot/vmlinuz-2.4.18-3 ro root=/dev/hda1

title Otro_S.O.
    rootnoverify (hd0,2)
    chainloader +1
```

entregando información similar a la de lilo y se carga ejecutando:
> grub-install /dev/XXX

donde XXX corresponde al disco(u otro dispositivo) donde queremos escribir la información de booteo. Por la configuración de tales archivos no deben preocuparse pues la instalación de un kernel los actualiza debidamente.

4.2. Niveles de ejecución

Luego de lo anterior entra en acción el kernel de **Linux**, que permite a los programas interactuar con el hardware, encargándose además del sistema de archivos, los procesos y la comunicación.

Una vez cargado el kernel, lo primero que éste hace es buscar un programa `init` a ejecutar, específicamente `/sbin/init`. El trabajo del programa `init` es lanzar a todo el resto de los programas para que el sistema quede funcionando, por ejemplo, revisar el sistema de archivos y montarlo, lanzar los procesos(daemons) encargados de logs de sistema, red, servicios, control del mouse, etc. También ejecuta los procesos `getty` que permiten el login en los terminales virtuales. `init` puede cargar el sistema en distintos niveles, para saber qué nivel usar lee del archivo `/etc/inittab` una línea que dice:

```
id:5:initdefault:
```

En este caso, nivel 5. Los niveles que existen son los siguientes:

- 0** - Halt (detener)
- 1** - Modo mono-usuario
- 2** - Modo Multiusuario, sin NFS(network file system)
- 3** - Modo Full multiusuario
- 4** - Sin uso
- 5** - Nivel 3 + Ambiente Gráfico(X11)
- 6** - Reboot (reiniciar)

Para cambiar de un nivel a otro se puede ejecutar: > `init <número>`

donde `<número>` corresponde al nivel que se quiere cambiar.

4.3. Scripts de arranque y procesos

Luego de lo anterior `init` ejecuta el script de inicialización del sistema ubicado en `/etc/rc.d/rc.sysinit`. Una vez concluido, es hora de lanzar el resto de los procesos. Para esto, existe una estructura de archivos y directorios ubicada en `/etc/rc.d/`. Sobre ese directorio existen una serie de subdirectorios `/etc/rc.d/rcX.d` donde la X identifica el nivel 0, 1, 2, 3, 4, 5 y 6.

Al interior de estos directorios existen links a scripts ubicados en `/etc/init.d/` que corresponden a los procesos a ejecutar. El nombre de los links tiene una función especial, que es identificar qué proceso debe correr primero que otro y qué procesos deben detenerse dependiendo del nivel.

Por ejemplo algunos archivos de `/etc/rc.d/rc3.d`:

```
lrwxrwxrwx 1 root root 18 Mar 4 15:45 K92iptables -> ../init.d/iptables
lrwxrwxrwx 1 root root 15 Mar 4 15:45 K95kudzu -> ../init.d/kudzu
lrwxrwxrwx 1 root root 17 Apr 19 15:48 S10network -> ../init.d/network
lrwxrwxrwx 1 root root 16 Mar 4 12:30 S12syslog -> ../init.d/syslog
```

que representan lanzar el proceso(S) o matar(K) el proceso en el orden que indican los números 92, 95, 10 y 12. La idea es que cada nivel corra sus determinados procesos, por lo que al cambiar de nivel se deban terminar los procesos que no formen parte del nivel e iniciar los que sí. Se puede ver también que todos los archivos son links a los scripts ubicados en `/etc/init.d` que corresponden a los daemons del sistema.

Para personalizar los niveles se deben crear y eliminar los links respectivos, sin embargo existe una herramienta que permite hacer lo mismo de una manera mucho más simple, `chkconfig`. Por ejemplo las instrucciones:

```
> chkconfig --level 35 proceso1 on
> chkconfig --level 123 proceso2 off
```

indicarían ejecutar el `proceso1` cuando se cargue el nivel 3 o el 5, y terminar el `proceso2` cuando se carguen los niveles 1, 2 o 3. Se puede obtener una lista de la configuración de los daemons con: `> chkconfig --list`

Las últimas tareas de `init` son lanzar gettys en modo `respawn`, o sea, esto significa que existirán consolas virtuales, y que además, cuando por alguna razón estos procesos terminen, `init` los llamará nuevamente.

Al terminar la carga del sistema, nos logueamos y ejecutamos:

```
> ps aux
```

USER	PID	%CPU	%MEM	SIZE	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.1	8.0	1284	536	? S		07:37	0:04	init [2]
root	2	0.0	0.0	0	0	? SW		07:37	0:00	(kflushd)
root	3	0.0	0.0	0	0	? SW		07:37	0:00	(kupdate)
root	4	0.0	0.0	0	0	? SW		07:37	0:00	(kpiod)
root	5	0.0	0.0	0	0	? SW		07:37	0:00	(kswapd)
root	52	0.0	10.7	1552	716	? S		07:38	0:01	syslogd -m 0
root	54	0.0	7.1	1276	480	? S		07:38	0:00	klogd
root	56	0.3	17.3	2232	1156	1 S		07:38	0:13	-bash
root	57	0.0	7.1	1272	480	2 S		07:38	0:01	/sbin/agetty 38400 tt
root	64	0.1	7.2	1272	484	S1 S		08:16	0:01	/sbin/agetty -L ttyS1
root	70	0.0	10.6	1472	708	1 R		Sep 11	0:01	ps aux

Obtenemos la lista de procesos corriendo en el sistema. La información viene del directorio `/proc`. Nótese como el número identificador del proceso(PID) de `init` es 1, o sea, el primero. Los procesos 2, 3, 4 y 5 pertenecen al kernel y no muestran información de memoria ocupada porque esa información es confidencial al kernel. Los paréntesis indican que no se tiene información tampoco del comando utilizado para llamar al proceso. Luego vienen los daemons de log del sistema que escriben información de lo que va aconteciendo en archivos ubicados en `/var/log`. Al final en la lista, los procesos gettys que nos entregan una consola virtual para loguearnos y obtener una shell o intérprete de comandos, en este caso, `bash`.

Una vez que hayamos concluido nuestras tareas, existen varias formas de apagar el equipo o reiniciarlo.

Apagar halt, poweroff, shutdown -h now, init 0

Reiniciar reboot, shutdown -r now, init 6

Estos comandos llevarán a cabo el proceso inverso al encendido, terminando los procesos activos, desmontando el sistema de archivos y finalizando el sistema.

4.4. El kernel en Linux

El kernel (núcleo) del sistema es su parte medular, que interactúa directamente con la mayor parte de los dispositivos y ofrece las abstracciones familiares como archivos y procesos. Como tal, es una parte crítica del sistema, y también la que más íntimamente depende del hardware instalado en la máquina. El paquete del núcleo en esta máquina tiene 30 MiB de archivos a instalar, lo cual sería imposible de acomodar en una máquina razonable. Y precisamente una de las ventajas de Linux es que es capaz de correr en máquinas muy limitadas.

Como una forma de resolver el problema que significaría el tener que tener un núcleo capaz de manejar directamente la enorme variedad de configuraciones que Linux soporta, se inventó la idea de manejar *módulos*, piezas del núcleo que se agregan al sistema en funcionamiento. Núcleos modernos incluso son capaces de cargar módulos según demanda, basta hacer referencia a la funcionalidad requerida (un dispositivo, un sistema de archivos, e incluso manejo de protocolos) para que los módulos requeridos se carguen automáticamente.

4.4.1. Comandos de manejo de módulos

Para manejar módulos hay varios comandos, de los cuales trataremos sólo los de más alto nivel. Hay comandos adicionales, pero son de interés sólo de desarrolladores del núcleo mismo.

```
lsmod
```

Muestra los módulos actualmente cargados en el núcleo. Da el nombre de cada uno, su tamaño, el número de usuarios directos, si está o no en uso (o está sujeto a ser eliminado automáticamente), y la lista de módulos que dependen de él actualmente (esto es independiente de los usuarios directos).

```
modinfo módulo...
```

Muestra información general sobre el módulo, como el nombre del archivo, una descripción somera, autor, licencia, y los parámetros que el módulo acepta con sus tipos.

```
depmod [-ae] [versión]
```

Los módulos pueden depender unos de otros, este comando construye el archivo `modules.dep` que usa `modprobe(8)` para cargarlos en el orden adecuado. La opción `-a` especifica ubicar módulos en todos los directorios mencionados en el archivo `/etc/modules.conf` (`modules.conf(5)`). La opción `-e` solicita se muestren todos los símbolos que causan errores en el proceso. De no darse versión del núcleo, procesa los módulos para el núcleo que está corriendo. Red Hat corre este comando desde `/etc/rc.sysinit`, cuando el sistema se inicia.

```
modprobe módulo...
```

Carga los módulos indicados (y todos los módulos que requieran, según indicado en `modules.dep`) y los inicializa.

```
rmmod [-ar] [módulo...]
```

Descarga los módulos indicados, si están libres. Con `-r` descarga los módulos de los que dependen también, en caso que queden libres. La opción `-a` especifica eliminar todos los módulos en desuso, aunque esto no es completamente funcional.

```
/etc/modules.conf
```

El archivo central de configuración del sistema de módulos, El uso principal es asociar módulos específicos a funcionalidades requeridas. Por ejemplo, para asociar la interfaz de red `eth0` al módulo `eeepro100` se ingresa:

```
alias eth0 eeepro100
```

Se pueden asociar parámetros a un módulo o a un alias, de forma que al cargarse el módulo tome esos argumentos. Para una tarjeta de red NE 2000 ISA podría ser:

```
alias eth1 ne
options eth1 io=0x320 irq=11
```

También se pueden especificar comandos a ejecutar antes o después de cargar o descargar un módulo dado:

```
pre-install modulo comando
post-install modulo comando
pre-remove modulo comando
pre-install modulo comando
```

4.4.2. El sistema de archivos inicial

Es posible que el funcionamiento del sistema requiera módulos que no son de uso corriente, como es el caso de sistemas con discos SCSI. En tal caso se requiere cargar los módulos requeridos incluso antes de comenzar a usar archivos en el disco. Esto se resuelve mediante el mecanismo de un sistema de archivos mínimo (`initrd`) que se carga en memoria junto con el núcleo. Inicialmente el núcleo usa este sistema de archivos inicial, cargando los módulos requeridos de allí, para luego liberarlo y montar los sistemas de archivos del disco.

```
mkinitrd imagen versión
```

Crea el archivo que contiene la `imagen` comprimida del sistema de archivos inicial para el núcleo de la `versión` dada. Red Hat coloca la imagen para el núcleo `versión` en `/boot/initrd-versión.img`.

4.5. Gestión Sistemas de Archivos

4.5.1. Sistema de archivo de Linux

Linux organiza la información en archivos, los cuales están contenidos en directorios. Un directorio puede contener subdirectorios, teniendo así una estructura jerárquica, como en cualquier otro sistema operativo.

Las nuevas versiones de Linux (incluido Red Hat) siguen el estándar FSSTND (Linux Filesystem Standard), el cual estipula los nombres, la ubicación y la función de la mayoría de los directorios y los archivos del sistema. Conociendo esta estructura básica, el usuario/administrador podrá moverse más fácilmente por los directorios, ya que la mayoría de éstos tienen un uso específico o determinado.

Algo importante que conviene recordar cuando se examina el sistema de archivos, es el hecho de que todo el sistema está incluido bajo un gran árbol de directorios, que se extiende por una o más Unidades de Disco, CDROM, etc. Esto significa que para acceder a las diferentes unidades no se hace a través de una letra como en DOS o Windows. En su lugar, todas se montan como subdirectorios que residen bajo el directorio raíz (/). En esencia cada parte del sistema (incluyendo el hardware del computador) reside en el sistema de archivos y se accede a ellos como archivos. Por supuesto no son archivos corrientes como los de texto, pero permiten hacer que el control a su acceso sea una tarea más simple. Entre los directorios más comunes encontramos:

Directorio	Descripción
/	Raíz (root), forma la base del sistema de archivos.
/boot	Archivos del kernel como la imagen, un mapa de símbolos, archivos a leer en el proceso de boot.
/bin	Archivos ejecutables esenciales para todos los usuarios.
/dev	Archivos de dispositivos. Descrito más adelante.
/etc	Archivos de configuración.
/etc/rc.d	Archivos de inicialización.
/home	Generalmente, directorios de los usuarios.
/lib	Bibliotecas esenciales y módulos del kernel.
/mnt	Directorios donde montar diversos dispositivos temporalmente.
/mnt/cdrom	Directorio donde se acostumbra montar el CD-ROM.
/mnt/floppy	Directorio donde se acostumbra montar el diskette.
/opt	Directorio opcional, para almacenar o instalar paquetes extra.
/proc	Información sobre el sistema. Descrito más adelante.
/root	Directorio del usuario principal del sistema.
/sbin	Archivos ejecutables para tareas de administración.
/tmp	Temporal.
/usr	Programas, documentación, fuentes, etc.
/var	Archivos variables del sistema, logs, correo, etc.

Algunos subdirectorios de /usr:

Directorio	Descripción
/usr/X11R6	Paquete XFree86 (X-Windows).
/usr/bin	Archivos ejecutables para usuarios.
/usr/share/doc	Documentación.
/usr/include	Archivos de encabezado(bibliotecas).
/usr/share/info	Sistema de información GNU info.
/usr/lib	Bibliotecas.
/usr/local	Archivos de programas que no son parte de la distribución.
/usr/share/man	Manuales.
/usr/sbin	Archivos ejecutables de administración.
/usr/src/redhat	Código fuente.
/usr/src/linux	Código fuente del kernel de \Linux.

Algunos subdirectorios de /var:

Directorio	Descripción
/var/lib	Información del estado de aplicaciones.
/var/local	Variables de aplicaciones en /usr/local.

/var/lock	Archivos de lock sobre archivos.
/var/log	Registros del sistema.
/var/named	Archivos del DNS.
/var/yp	Base de datos para NIS (Network Inf. Service).
/var/preserve	Archivos de respaldo después de una caída para vi o ex.
/var/run	Archivos relevantes a programas corriendo.
/var/spool	Colas de trabajos para realizar mas tarde.
/var/spool/at	Archivos creados por comando at.
/var/spool/cron	Archivos creados por comando crontab.
/var/spool/lpd	Archivos de impresora.
/var/spool/mail	Archivos de correo de cada usuario.
/var/spool/mqueue	Archivos de correo de salida.
/var/spool/news	Archivos de noticias de salida.
/var/tmp	Temporal.

4.5.2. Permisos de archivos y directorios

Mención especial merecen los archivos bajo el directorio /dev. Encontramos dos categorías importantes, los dispositivos *de bloque* y los de *caracter*.

Los primeros corresponden a dispositivos que almacenan datos, por ejemplo discos duros, mientras los segundos están asociados a dispositivos de transferencia de datos, por ejemplo, el mouse.

Ejemplo:

```
brw-rw---- 1 root    disk      3,   1 Aug 30 2001 /dev/hda1
crw----- 1 root    root      10,  1 Jul 11 11:49 /dev/psaux
```

que corresponden a la primera partición del disco primario maestro y al mouse, en este caso PS/2.

La primera letra de la línea en la sección permisos identifica el tipo de dispositivo (Toda esa información se explica más adelante en detalle). Lo que importa ahora, son los dos números que aparecen antes de la fecha.

El primero corresponde al Major number y el segundo al Minor number, ambos conocidos como números mágicos permiten al sistema identificar el dispositivo al que están relacionados.

En el ejemplo anterior:

Major	Descripción	Minor	Descripcion
3	IDE primario	1	Primera partición
10	Dispositivo de mouse no serial	1	PS/2

Algunos otros dispositivos importantes son:

/dev/hd*	Discos IDE
/dev/sc*	Discos SCSI
/dev/fd*	Floppy
/dev/tty*	Puertas seriales
/dev/pty*	Pseudo terminales

Sobre el directorio /proc existen archivos de información que en su mayoría están vacíos, lo que ocurre es que al leerlos, el kernel los escribe con información del momento, por ejemplo:

```
> ls -l /proc/uptime
```

```
-r--r--r-- 1 root    root      0 Jul 11 16:02 uptime
```

el archivo pareciera estar vacío, sin embargo al ver lo que contiene con: `> cat /proc/uptime` 512420.18 510135.35

el kernel nos indica que el computador lleva encendido 512420 segundos aproximadamente.

Dado que Linux es un sistema multiusuario enfocado a redes, el sistema de archivos provee un sistema de permisos para evitar, por ejemplo, manipulación por otros usuarios de archivos ajenos. Asimismo, este sistema dispone de modos para que archivos sean compartidos por un mismo grupo de usuarios. Los permisos de un archivo se pueden ver mediante la ejecución del comando de listado, `ls`, con un modificador para que nos entregue mayor información:

```
> ls -l
```

```
-rw-r--r-- 1 snoopy users 3147 Dec 26 1993 README
```

En este ejemplo el dueño del fichero que se llama README es *snoopy*. El grupo de usuarios al que pertenece es *users*. La fecha de su creación es el 26 de diciembre de 1993. El tamaño es de 3147 bytes. Las 10 letras y guiones iniciales son los permisos y se interpretan de la siguiente forma:

(dir)(rd,wd,ed)(rg,wg,eg)(ro,wo,eo)

(dir) :

- “d” si es un directorio,
- “l” si es un link,
- “b” si es un dispositivo de bloque,
- “c” un dispositivo de caracteres,
- “p” si es una tubería (pipe),
- “s” si es un socket y
- “-” si es un archivo ordinario.

(rd,wd,ed) : valen 'r','w','x' si *el dueño* tiene permiso de lectura, escritura o ejecución sobre el fichero, respectivamente.

(rg,wg,eg) : valen 'r','w','x' si *el grupo* tiene permiso de lectura, escritura o ejecución sobre el fichero, respectivamente.

(ro,wo,eo) : valen 'r','w','x' si *los demás* tiene permiso de lectura, escritura o ejecución sobre el fichero, respectivamente.

Un guión en cualquier posición, indica que no se tiene el permiso correspondiente. Sobre directorios, la interpretación de los permisos es:

- “x” permite acceder al directorio,
- “r” permite listar el directorio y
- “w” permite agregar o borrar archivos del directorio.

Existen además cuatro permisos especiales, que son los siguientes:

SUID : El SUID significa 'Set Users ID', o sea, “Fijar la identificación del dueño” y el programa se ejecuta con la identificación y los permisos del dueño. Un caso típico es el del programa `passwd`, que cambia la contraseña de entrada de los usuarios modificando el archivo `/etc/passwd`. Para que un usuario normal no pueda modificar el archivo `/etc/passwd` se le quitan los permisos correspondientes, se activa el SUID sobre el programa `passwd` y éste, como se ejecuta con los permisos `root`, sí puede modificar el archivo y lo modifica sólo en la sección que contiene información del usuario que ejecutó el comando.

SGID : El SGID es análogo y significa 'Set Group ID', es decir, "Fijar la Identidad del grupo". Lo que provoca es que cuando un usuario ejecute un programa con el SGID activo, este se va a ejecutar con la identidad (y se supone, los privilegios) del grupo al que pertenece el archivo.

Locking Bit : Lo que hace es permitir que un programa cierre el acceso a un archivo, evitando así escrituras simultáneas que conlleven una corrupción del fichero.

Sticky Bit : Este bit, al contrario que los otros, en los que lo pueden activar el dueño, o cualquiera con permisos para ello, sólo lo puede activar root. Provoca que el programa al que se le aplica quede residente en memoria de forma que la próxima vez que sea llamado, su carga sea más rápida. Cuando se le aplica a un directorio, impide que nadie más que su dueño puede renombrar o borrar algún archivo del mismo. En realidad no suponen ninguna adición, sino que son casos especiales de los permisos normales.

Los permisos son dependientes entre directorios y archivos, por ejemplo, un directorio que no tenga permisos de lectura, escritura y ejecución para mi grupo impedirá que los archivos que están dentro sean leídos, modificados o ejecutados por mi grupo, aunque tengan los permisos necesarios.

El comando para modificar los permisos es `chmod` y tiene la siguiente sintaxis:

```
> chmod <permisos> <archivo>
```

Por ejemplo se desea que todos las personas puedan ver y escribir sobre el archivo `curso.tex`, entonces ejecutamos:

```
> chmod a+rw creditos.tex
```

o su equivalente en números:

```
> chmod 666 creditos.tex
```

Básicamente la nomenclatura del comando es la siguiente:

```
chmod <a,u,g,o><+.-><rxw> <archivo>
```

`e` indica dar(+) o quitar(-) los permisos de lectura(`r`), escritura(`w`) o ejecución(`x`) a todos(`a`), el dueño(`u`), el grupo(`g`) o los demás(`o`).

En números funciona de la siguiente manera, el permiso de lectura tiene valor 4, el de escritura 2 y el de ejecución 1. Luego cada grupo de permisos (dueño, grupo y otros) tiene asignado un número que corresponde a la suma de los permisos. Es decir:

```
> chmod 764 archivo
```

implica todos los permisos(7) para el dueño, lectura y escritura(6) para el grupo y sólo de lectura(4) para los demás.

El comando para modificar el dueño y grupo de un archivo es `chown` y tiene la siguiente sintaxis:

```
> chown <user>.<group> <archivo>
```

Lo anterior cambia al dueño y grupo del archivo actual por dueño `user` y grupo `group`.

4.5.3. Montaje y desmontaje de sistemas de archivos

Ya se ha visto que Linux accede a los dispositivos mediante archivos (directorio `/dev`), y por este motivo, no existe el concepto de unidades, ya que todo está bajo el directorio principal `/`. Por ejemplo no se accede a la primera disquetera mediante la orden `A:` como en DOS sino que se debe montar. De este modo, tenemos dos conceptos nuevos:

montar : Decirle a Linux que se va a utilizar un determinado dispositivo, con un determinado sistema de archivos y se accederá a éste en un directorio especificado.

desmontar : Decirle a Linux que se ha dejado de utilizar un determinado dispositivo y liberar el directorio utilizado.

Para montar un determinado sistema de archivos de un dispositivo, se utiliza el comando mount.

La sintaxis es la siguiente:

```
> mount -t <tipo_fs> <dev> <dir> -o <opciones>
```

donde tipo_fs puede ser cualquiera de los que aparece en la lista;

Tipo	Descripción
ext2	Sistema de archivos de Linux.
msdos	Sistema de archivos de DOS.
vfat	Sistema de archivos de Windows 9X (nombres largos).
iso9660	Sistema de archivos de CD-ROM.
nfs	Sistema de archivos compartido por red (exportado).

dev puede ser cualquier dispositivo del directorio /dev o, en el caso de nfs, un sistema de archivos de otro computador; dir es el directorio donde se accederá el dispositivo y opciones pueden ser cualquiera de las que se nombran a continuación:

Opción	Descripción
rw	Lectura/escritura.
ro	Sólo lectura.
exec	Se permite ejecución.
user	Los usuarios pueden montar/desmontar.
suid	Tiene efecto los identificadores de propietario y del grupo.
auto	Se puede montar automáticamente.
async	Modo asíncrono.
sync	Modo síncrono.
dev	Supone que es un dispositivo de caracteres o bloques.

En el caso de no poner ninguna opción, mount utilizará las opciones por defecto. Una vez montado el dispositivo, si no se va a volver utilizar se puede desmontar con el comando umount como sigue:

```
> umount <dir>
```

Siempre después de utilizar un dispositivo hay que desmontarlo, para que se almacenen correctamente los datos en dicho dispositivo.

Un ejemplo de ello, es el hecho de que, un lector de CD-ROM, que haya sido montado no se abrirá hasta que no se desmonte. O los datos copiados a un diskette, no estarán realmente copiados hasta que se desmonte el dispositivo o se llame al comando sync.

Ejemplos:

Diskette de DOS:

```
> mount -t msdos /dev/fd0 /mnt/floppy -o rw,noexec
> umount /mnt/floppy
```

Diskette de Windows 9X:

```
> mount -t vfat /dev/fd0 /mnt/floppy -o user,rw
> umount /mnt/floppy
```

CD-ROM:

```
> mount -t iso9660 /dev/cdrom /mnt/cdrom -o ro
> umount /mnt/cdrom
```

Directorio exportado desde pc2:

```
> mount -t nfs host2:/tmp /mnt/net
> umount /mnt/net
```

4.5.3.1. Archivo /etc/fstab

En ocasiones, cuando se tienen varios dispositivos que se suelen montar, se puede ahorrar tener que escribir continuamente la orden mount, simplemente incluyendo una línea en el archivo /etc/fstab. Este archivo contiene líneas donde se indica qué dispositivo se debe montar, el lugar dónde montarlo, así como el sistema de archivos y las opciones (en este archivo, se pueden poner dos opciones más: auto y noauto, que indican si se debe montar automáticamente al arrancar el sistema o no, respectivamente).

Un ejemplo de /etc/fstab puede ser:

Dispositivo	Directorio	FS	Opciones		
/dev/hda1	/	ext2	defaults	1	1
/dev/hda2	/home	ext2	defaults	1	2
/dev/hda3	/tmp	ext2	defaults,noexec	1	2
/dev/hda4	none	swap	defaults	1	2
none	/proc	proc	defaults	0	0
/dev/fd0	/mnt/floppy	ext2	noauto,user,noexec,rw	0	0
/dev/fd0	/mnt/msdos	vfat	noauto,user,noexec,rw	0	0
/dev/cdrom	/mnt/cdrom	iso9660	noauto,user,noexec,ro	0	0
/dev/sda4	/mnt/iomegazip	vfat	noauto,user,noexec,rw	0	0
host2:/tmp	/mnt/net	nfs	defaults	0	0

Con un archivo /etc/fstab como el anterior, cualquier usuario podría hacer:

```
> mount /mnt/msdos
> umount /mnt/msdos
```

para montar y desmontar un diskette respectivamente. Sin embargo, sólo el administrador podría montar y desmontar el directorio /mnt/net.

Sobre los últimos dos números que aparecen, estos indican que sistemas de archivos se deben respaldar si se llama al comando dump (1 para respaldar, 0 sino) y en qué orden se deben revisar las particiones al inicio del sistema (0 para no revisar, 1 para la partición raíz y 2 para las demás).

4.5.3.2. Uso de mtools

El hecho de tener que montar y desmontar puede ser un poco engorroso a la hora de utilizar determinados dispositivos (comúnmente, la diskettera). Por ello, se dispone de las herramientas *mtools*. Dichas herramientas, utilizan los dispositivos sin tener que montar y desmontar; y su sintaxis es parecida a la de los programas de DOS.

Comando	Descripción
<code>mdir</code>	Muestra el contenido del dispositivo <code>dir</code> .
<code>mcopy</code>	Copia archivos <code>copy</code> .
<code>mdel</code>	Borra archivos <code>del</code> .
<code>mformat</code>	Formatea la unidad <code>format</code> .
<code>mcd</code>	Cambia de directorio <code>cd</code> .
<code>mmd</code>	Crea un directorio <code>md</code> .
<code>mrd</code>	Borra un directorio <code>rd</code> .

La referencia a la disquetera es como unidad `a:`: por ejemplo:

```
> mcopy <archivo> a:  
> mdir a:
```

copiará el archivo al diskette y listará lo que hay en el diskette.

4.5.4. Sistemas de archivos en red

Existe una modalidad de archivos que se pueden compartir a través de la red, a través del protocolo de NFS (Network File System). La finalidad del NFS, es poder montar en localmente, un directorio desde otro lugar de la red y verlo como si estuviese de manera local.

Para poder exportar archivos de NFS, es necesario en el servidor, configurar un archivo ubicado en el archivo `/etc/exports` e iniciar el servicio `nfs`. En el cliente es necesario tener funcionando el servicio `autofs` y montar el directorio exportado.

4.5.4.1. El archivo `/etc/exports`

En este archivo se especificará qué directorio (o directorios) se puede compartir a otras redes y con qué atributos se compartirán; existe una página de manual que describe sus detalles, pero en general, la sintaxis del archivo es la siguiente:

```
<Directorio1> <opciones dir1>  
<Directorio2> <opciones dir2>  
  
<DirectorioN> <opciones dirN>
```

Donde podemos notar que se pueden compartir `N` directorios, con permisos distintos y a distintas redes.

Algunas de las opciones más comunes son:

rw Permite escritura dentro del directorio a ambos lados, o sea, tanto en el servidor, como en el cliente.

ro Describe que el directorio que se está exportando, sólo podrá ser leído en el cliente y no podrá modificarlo.

secure Se indica que el puerto de transmisión de datos, será uno menor al 1024, o sea, que pertenece a uno de los puertos de servicios.

sync Establece que mientras se está escribiendo en el cliente, se está escribiendo al mismo tiempo en el servidor. Puede aumentar el Overlay de la red, disminuyendo su ancho de banda efectivo, pero es más seguro, ya que si el cliente llegase a tener algún problema, se asegura que en el servidor la información ya estará escrita.

Algunos ejemplos de este archivo pueden ser:

```
/home/snoopy    192.168.0.0/255.255.0.0
/pub           (ro,insecure,all_squash)
```

En la primera línea, se está exportando el directorio `/home/snoopy` a toda la red `192.168.0.0`, la cual tiene máscara `255.255.0.0`. Como se verá más adelante, en la sección de redes, podría haberse puesto `/16`

En la segunda línea, se está exportando un directorio público de un ftp a todos los posibles hosts existentes, ejecutando todas las transacciones sobre la cuenta `nobody`. La opción “`insecure`” en esta línea, nos dice que no se usen los puertos reservados para los servicios, como el NFS (o sea, que pertenezca a los puertos del 1024 al 65536).

4.5.4.2. `/etc.init.d/autofs`

El servicio `autofs`, es el que permite a un cliente, montar desde un servidor de NFS un directorio que está siendo exportado.

Internamente, `autofs` se encarga de ejecutar el demonio `automount`, el cual consultará los archivos de `/etc/auto.master`, para encontrar los puntos de montaje que iniciará `autofs` con sus parámetros apropiados.

Para ocupar este servicio, podemos llamarlo de dos maneras:

```
> /etc/init.d/autofs <start | stop | reload >
> service autofs <start | stop | reload >
```

Donde le pedimos que inicie (`start`), termine (`stop`), o se relea el archivo (`reload`).

4.5.4.3. `/etc/init.d/nfs`

Una vez configurado el archivo de configuración `/etc/exports`, es necesario iniciar el servicio de `nfs` en el servidor. Para ello, utilizaremos una sintaxis similar a la de `autofs`, para llamar al servicio. Esta puede ser:

```
> /etc/init.d/nfs <start | stop | status | restart | reload>
> service nfs <start | stop | status | restart | reload>
```

Donde:

start Inicia el servicio, llamando los archivos de configuración de `/etc/exports`

stop Detiene el servicio.

status Consulta cuál es el estado actual del servicio

restart Reinicia el servicio, primero bajándolo y a continuación volviéndolo a subir.

reload Reinicia el servicio, releyendo el archivo `/etc/exports` y aplicando los cambios sobre las diferencias que existan en el estado actual.

4.5.5. Verificación de sistema de archivo

Sobre errores en el disco, sistemas anteriores usaban particiones del tipo `ext2` que necesitaban verificarse y repararse a mano. Cuando el sistema se caía por un corte de luz, al volver a encenderse verificaba las particiones y en caso de encontrar errores solicitaba la intervención del super-usuario, para que ejecutara las herramientas de reparación de disco como `fsck`:

```
> fsck /dev/hda1
```

con esto se chequeaban los i-nodos, archivos, etc. de la partición y si algún error no se podía reparar, se copiaba lo que quedaba de información en ese sector al directorio `/lost+found` que existe en cada partición para su posterior recuperación.

El Redhat 9 trae un sistema de archivo de tipo ext3 que maneja automáticamente errores en los discos por causa por ejemplo, de un apagado abrupto, este sistema se llama *Journalism*.

4.5.6. Gestión de Swap

Por último, debemos mencionar la gestión de SWAP, espacio especial reservado del disco a usar por el kernel como memoria temporal cuando la RAM disponible se empiece a agotar. Supuestamente la partición inicial que creamos de tipo SWAP debiera ser suficiente, sin embargo, si llegaran a aumentar los requerimientos del sistema, podemos crear archivos de swap.

Para crear un archivo de un tamaño específico con puros ceros podemos usar:

```
> dd if=/dev/zero of=archivo_swap bs=1024 count=100
```

con lo anterior creamos un archivo de 100×1024 bytes, es decir, 100 kilobytes vacíos (lleno de ceros). Ahora necesitamos darle un formato al archivo, para eso usamos:

```
> mkswap archivo_swap
```

Lo único que falta es habilitar el archivo con:

```
> swapon archivo_swap
```

Para saber si esta funcionando podemos mirar la información bajo `/proc`:

```
> cat /proc/swaps
```

Filename	Type	Size	Used	Priority
/dev/hda3	partition	265064	42716	-1
/root/archivo_swap	file	96	0	-2

5. Administración del Sistema

5.1. Gestión de Usuarios y Grupos

5.1.1. Perfiles de usuario

Cuando un computador es usado por varias personas se hace necesario diferenciar a los usuarios para, por ejemplo, poder proteger la privacidad de los archivos de uno con respecto a otro. Es por esto que cada usuario recibe un nombre único para acceder al sistema. El usuario posee más atributos que sólo su nombre, tiene además un cierto espacio en el disco, una clave, recursos, etc.

El kernel de Linux trata a los usuarios como simples números, cada uno posee un único número identificador conocido como el *uid*, esto porque para un computador es más fácil procesar números que palabras. Una base de datos externa al kernel asigna los nombres a cada identificador. Posee además el resto de la información del usuario.

Además existe el concepto de grupo, que permite juntar varios usuarios con alguna característica en común, como por ejemplo, el acceso a un recurso. De esta forma sólo habilitamos el recurso a un determinado grupo, evitándonos habilitar a cada uno por separado.

5.1.2. Herramientas para la gestión de usuarios

Para crear un usuario, tan sólo se necesita agregar información a la base de datos y crear un directorio en /home que será su espacio de trabajo. Se debe además configurar su entorno. Lo anterior se puede hacer manualmente o aprovechando herramientas que automatizan el proceso como por ejemplo:

```
> adduser <nombre_usuario>
```

que agregará al usuario a la base de datos sin más información que su nombre y además creará y configurará su espacio de trabajo. Un ejemplo más elaborado podría ser:

```
> adduser <nombre_usuario> -c "Información del Usuario" -u <uid>
```

que agrega información del usuario y le asigna un número identificador.

La base de datos de la que hablamos corresponde a archivos de texto ubicados en el directorio /etc. El primero de ellos es passwd que contiene la siguiente información:

```
user:x:100:100:Usuario Uno:/home/user:/bin/bash
```

que indica nombre de usuario, un espacio(x) para la clave que se almacena en otro archivo, uid, gid, nombre completo o descripción, el directorio de trabajo, y la shell o interprete de comandos.

La clave que nombramos se guarda encriptada en el archivo shadow que luce como sigue:

```
user:$1$Wikjh$jh4$jeB0bsd4zvGMW/U0jXz0:11751:0:99999:7:::
```

indica nombre de usuario, clave encriptada, días desde el primero de enero de 1970 que no se ha cambiado la clave, mínimo de días antes de que se permita cambio de clave, máximo de días sin cambiar la clave, días de advertencia de expiración de la clave, días después de expiración que la cuenta se deshabilita, días desde el primero de enero de 1970 cuando la cuenta será deshabilitada y un campo sin uso.

Sobre el número de usuario y grupo, éste se asigna por defecto al primer número disponible después del último ocupado, o se puede elegir manualmente, lo importante es que se mantenga a lo largo de todos los computadores que un usuario utilice.

5.1.2.1. El directorio /etc/skel

La configuración inicial de la creación de un usuario, se obtiene de /etc/skel, un directorio que posee archivos de configuración básicos necesarios para empezar a trabajar en el sistema.

Por ejemplo:

.bashrc para configurar la shell.

.bash_profile para ejecutar comandos al inicio de sesión.

.bash_logout para ejecutar comandos al final de la sesión.

5.1.2.2. Otros comandos útiles

Existen algunos comandos para cambiar características de una cuenta.

Por ejemplo:

chfn Cambia el campo de nombre o descripción de la cuenta.

chsh Cambia la shell a ocupar.

passwd Cambia la clave del usuario (se debe usar luego de crear un usuario).

Para eliminar a un usuario se puede usar el comando:

```
> userdel <nombre_usuario>
```

Nota: Es posible que no se eliminen todos los archivos de la cuenta, para esto podemos usar:

```
> find / -user <nombre_usuario>
```

que nos indicará todos los archivos de posesión de la cuenta que eliminamos y podremos borrarlos manualmente.

Si se quiere deshabilitar alguna cuenta por un tiempo podemos, por ejemplo, editar el archivo /etc/shadow y modificar la clave encriptada con cualquier frase:

```
user:* cuenta cerrada *:11751:0:99999:7:::
```

Como el password se guarda encriptado, no hay problema con poner una frase tan simple. Cuando la cuenta se vuelva a habilitar sólo habrá que ejecutar el comando passwd para volver a asignar una clave. Otra forma, sería cambiando la shell por algún otro script que mostrara un mensaje, por ejemplo:

```
#!/usr/bin/tail -n 2
```

```
Esta cuenta ha sido cerrada por mal uso.
```

```
Póngase en contacto con el administrador llamando al 555444.
```

de esta forma, cuando el usuario ingrese al sistema, solo obtendrá un mensaje y no podrá hacer nada.

5.1.3. Grupos de usuario

Como ya explicamos, cada usuario pertenece a uno o más grupos. La única importancia real de las relaciones de grupo es la perteneciente a los permisos de archivos, cada archivo tiene un *grupo propietario* y un conjunto de permisos de grupo que define de qué forma pueden acceder al archivo los usuarios del grupo.

Hay varios grupos definidos en el sistema, como pueden ser bin, Mail, y sys. Los usuarios no deben pertenecer a ninguno de estos grupos; se utilizan para permisos de archivos del sistema. En su lugar, los usuarios deben pertenecer a un grupo individual, como users. Si se quiere ser detallista, se pueden mantener varios grupos de usuarios como por ejemplo estudiantes, soporte y facultad. El archivo `/etc/group` contiene información acerca de los grupos. El formato de cada línea es:

```
nombre de grupo:clave:GID:otros miembros
```

Algunos ejemplos de grupos pueden ser:

```
root:*:0:
usuarios:*:100:webmaster,snoopy,woodstock
invitados:*:200:
otros:*:250:charlie,lucy
```

El primer grupo, *root*, es un grupo especial del sistema reservado para la cuenta root. El siguiente grupo, *usuarios*, es para usuarios normales. Tiene un GID de 100. Los usuarios webmaster, snoopy y woodstock tienen acceso a este grupo.

Los usuarios pueden pertenecer a más de un grupo, añadiendo sus nombres de usuario a otras líneas de grupo en `/etc/group`. El comando `groups` lista a que grupos se tiene acceso.

El tercer grupo, *invitados*, es para usuarios invitados, y *otros* es para otros usuarios. Los usuarios charlie y lucy tienen acceso a este grupo.

Como se puede ver, el campo clave de `/etc/group` raramente se utiliza. A veces se utiliza para dar una clave para acceder a un grupo. Esto es raras veces necesario. Para evitar el que los usuarios cambien a grupos privilegiados (con el comando `newgroup`), se pone el campo de la clave a “*”.

Se pueden usar los comandos `addgroup` o `groupadd` para añadir grupos a su sistema. Normalmente es más sencillo añadir líneas a `/etc/group` uno mismo, puesto que no se necesitan más configuraciones para añadir un grupo. Para borrar un grupo, solo hay que borrar su entrada de `/etc/group`.

5.2. Administración de Paquetes

5.2.1. RedHat Package Manager (RPM)

La distribución Red Hat introdujo un sistema de administración de paquetes que hoy día es usado casi universalmente en el mundo Linux. La idea es guardar en una base de datos información sobre los archivos instalados en el sistema (a qué paquete pertenecen, tamaños y permisos, dependencias entre paquetes, entre otros). El comando `rpm(1)` tiene varios modos de operación, acá nos centraremos en la instalación, desinstalación, y algunas consultas. El nombre de un paquete tiene la forma siguiente:

```
nombre-version-release.arch.rpm
```

El **nombre** es el nombre del paquete, la **versión** es la versión base usada para construir el paquete, **release** se refiere a la versión modificada para distribución. La arquitectura **arch** identifica la máquina para la cual se creó el paquete, o **noarch** para paquetes que no dependen de la arquitectura (como documentación). Si aparece **src** acá, se trata de un paquete con fuentes para crear el paquete binario.

Los paquetes fuente contienen los fuentes originales del paquete, además de todas las modificaciones locales que aplicó la distribución, y archivos de configuración que controlan la construcción del paquete.

Se pueden encontrar paquetes adicionales a la distribución misma por ejemplo en <http://www.rpmfind.net>. En tal caso, debe tenerse cuidado de sólo instalar paquetes creados para la distribución exacta que se está usando. Distintas distribuciones tienen diferencias sutiles, con lo que paquetes extranjeros pueden producir problemas severos.

5.2.1.1. Consultas

```
rpm -q opciones de consulta
```

Las opciones de consulta incluyen `-f` (a qué paquete pertenece un archivo dado), `-l` (liste los archivos que pertenecen a un paquete dado), `-i` (dé información sobre el paquete), mientras `-v` solicita información adicional. El paquete puede darse como nombre de un paquete instalado, o con `-p` se da el archivo que lo contiene. Las opciones pueden combinarse, así `rpm -qfi /bin/bash` solicita información sobre el paquete que contiene el shell. Otras opciones importantes con `--whatprovides` (que paquete provee una funcionalidad particular, como un archivo dado) y `--whatrequires` (que paquete requiere alguna funcionalidad particular o depende de un paquete específico). Si está instalado el paquete con la base de datos completa de los paquetes de Red Hat (`rpmdb-redhat`), pueden usarse `--redhatprovides` y `--redhatrequires` para consultar la colección de paquetes de la distribución, no lo que está instalado.

5.2.1.2. Verificación

```
rpm -V opciones de verificación
```

Verifica lo que hay instalado contra la base de datos. Reporta archivos faltantes o que se hayan modificado en el paquete. Con `-Va` verifica todo (esto puede tomar largo tiempo).

5.2.1.3. Instalar

```
rpm -[iUF] opciones de instalación
```

Se pueden instalar (`-i`) nuevos paquetes, o actualizarlos (`-U`). La diferencia está en que al instalar queda la versión anterior, al actualizar ésta se elimina. La opción `--force` fuerza la instalación o actualización (para instalar una versión más antigua que la actual, o forzar reinstalar sobre la misma versión). Para simplificar la actualización está la opción de refrescar (`-F`), que actualiza sólo aquellos paquetes que ya están instalados. Opciones generales con `-v` (muestra las versiones de lo que se instala) y `-h` (muestra el avance del proceso). Los paquetes se pueden especificar como archivos locales, o como URLs a través de FTP o HTTP.

5.2.1.4. Desinstalar

```
rpm -e paquetes a desinstalar
```

Se indican los paquetes a desinstalar, ya sea con el nombre del paquete únicamente o la versión completa en caso que hayan varias versiones instaladas.

5.2.2. Actualización del sistema

El equipo de desarrollo de Red Hat Linux, al igual que en todas las distribuciones que existen en el mundo, constantemente están trabajando para desarrollar sistemas más seguros y estables, es por eso,

que cada vez que se encuentra un error, ya sea de seguridad (que es el más común), o de programación (algo que no funcione), se re-haga el paquete para que se actualice en su sistema.

La importancia de un sistema con los paquetes actualizados al día, es sumamente importante, aún si no se tiene el equipo conectado a una red. El malfuncionamiento de programas, y posibles errores a futuro, pueden ser prevenidos si tenemos el sistema al día; más aún si ha salido una nueva versión del Kernel o núcleo del sistema.

Por lo tanto, lo primero que vamos a hacer, una vez ya instalado el sistema, es proceder a updatear o upgradear nuestro equipo. Para ello podemos utilizar las herramientas *apt-get* o el mismo administrador de paquetes *RPM* que revisamos en la sección anterior.

5.2.2.1. Apt-get

APT - Advance Package Tool: Es un conjunto de herramientas que se utilizan para administrar paquetes de forma automatizada, de manera tal, que cuando el usuario solicita la instalación de un paquete (aplicación), el sistema también instala (o actualiza) todos los paquetes necesarios para el funcionamiento de esa aplicación (resolviendo dependencias).

Cabe señalar, que *apt-get* es un programa (paquete) que fue creado para la distribución Debian y debido a sus múltiples opciones y facilidad de uso se portó a RedHat, bajo el nombre de *apt-rpm*, pero por razones históricas, se mantuvo el nombre de la aplicación en *apt-get*. Debido a que fue creado para Debian, el manejo de paquetes lo hacía con el administrador de paquetes DEB. Ahora que ha sido portado a RedHat, trabaja sobre paquete RPM... Esto quiere decir, que *apt-get* es sólo una API o cáscara que trabaja sobre los paquetes RPM y no un manejador de paquetes distinto de él.

5.2.2.1.1. sources.list

Este archivo, ubicado en el directorio `/etc/apt` contiene la información de los servidores desde donde se traerán los paquetes.

Ejemplo: Archivo `sources.list`.

Todas las líneas que comiencen con `#` son comentarios.

```
#Package repository URLs
#Official repositories

#Directorios que contienen los paquetes actualizados del sistema.
rpm ftp://ftp.inf.utfsm.cl/pub/Linux/RedHat/apt-get/updates 9/redhat i386 i586

#Directorio de la distribución.
rpm ftp://ftp.inf.utfsm.cl/pub/Linux/RedHat/apt-get/rpms 9/redhat i386
```

5.2.2.1.2. Comandos de apt-get

Los comandos de *apt-get* siguen la siguiente estructura:

```
apt-get [opciones] comando
apt-get [opciones] install paquete [paquete... ]
```

La línea de comando puede ser una variación de los siguientes tipos básicos:

apt-get update Con este comando se actualizará la lista de paquetes que se encuentran en el servidor y serán bajados al computador local.

apt-get check Herramienta de diagnóstico, updatea el caché verificando integridad del sistema. Es recomendable ejecutarlo antes de empezar una actualización de la distribución

apt-get install algun_paquete Instala algún paquete nuevo, resolviendo dependencias automáticamente. Si el paquete *algun_paquete* ya está instalado, intentará actualizarlo.

apt-get upgrade Busca paquetes que estén desactualizados en el sistema y los actualiza automáticamente. Para actualizar el paquete y sus dependencias se debe utilizar el comando:

```
apt-get install paquete\_a\_actualizar
```

apt-get dist-upgrade Instala todos los paquetes básicos e intenta actualizar todo, instalando nuevos paquetes si es necesario. Esta es una manera más fácil de hacer una actualización de la distribución

apt-get remove algun_paquete Elimina el paquete *algun_paquete* y todos los demás paquetes que dependen de él.

apt-get clean Elimina los archivos que se encuentran en `/var/cache/apt` y que han sido bajados del servidor

6. Software de Productividad

6.1. Entornos de Escritorio

A continuación, veremos cómo funciona el entorno gráfico de Linux, el cual además posee diferentes “Escritorios”, que a su vez tienen distintas aplicaciones.

6.1.1. XFree86(1)

XFree86 es un conjunto de paquetes que dan soporte a las tarjetas de Video, esto es, hacen que el sistema operativo posea un entorno gráfico. Para poder instalar el sistema XWindow es necesario verificar que nuestro equipo tenga el Hardware apropiado, la memoria necesario y el espacio necesario en disco duro.

De espacio es necesario unos 50 ó 60MB en disco además de 16MB de memoria virtual⁷. Como se necesitan unos 4MB de RAM, entonces se debe reservar 12MB en el archivo swap. Cuanto más RAM posea el sistema, mayor será el rendimiento del sistema XFree86.

Por último, se necesitará una tarjeta de video que contenga un *chipset*⁸ de controlador de video soportado por XFree86. Para ello se puede consultarla página <http://www.xfree86.org/4.2.0/Status.html>, para una lista actualizada de los *chipset* soportados.

6.1.1.1. ¿Qué es XFree86?

XFree86 es una implementación de **X11** o el sistema **XWindow** y siguiendo la filosofía Linux, distribución libre y open-source.

XFree86 fue diseñado preferentemente para plataformas UNIX® y otros sistemas operativos basados en UNIX, tales como Linux, todas las variantes de BSD, Sun Solaris x86, Mac OS X (via Darwin). El resultado ha sido tan bueno como otras plataformas como OS/2 y Cygwin.

XFree86 proporciona una interfaz cliente/servidor entre el Hardware (ratón, teclado, etc.) y el ambiente de escritorio. Además, posee la infraestructura de “ventanas” y una interfaz de aplicaciones estándar (API). XFree86 es una plataforma independiente, extensible y además es una red transparente.

6.1.1.2. Configurando XFree86

Históricamente XFree86 ha sido una de las partes más complejas bajo Linux, en lo que respecta a su configuración. Este ya no es el caso para el hardware más habitual. Sin embargo, aún hay dos casos en los que la instalación puede ser difícil.

En primer lugar, el Hardware de más reciente aparición puede estar soportado por XFree86, o puede no estarlo en absoluto. Si XFree86 lo soporta, puede que tenga que usar versiones beta de XFree86, o incluso versiones parchadas del mismo. Éstas no estarán soportadas por las nuevas herramientas de configuración.

En segundo lugar, algunos proveedores no publican las especificaciones para sus tarjetas. Para que XFree86 soporte estas tarjetas, los desarrolladores deben efectuar una ingeniería inversa, lo que lleva mucho tiempo y esfuerzos. A menos que la tarjeta sea de uso extensísimo, puede que no haya soporte de XFree86 durante mucho tiempo. Hacer que una tarjeta sin soporte funcione puede resultarle difícil, cuando no imposible, si no es capaz de escribir usted mismo los controladores.

Éstos son sólo los peores casos. Para Hardware más extendido, le será suficiente usar *Xconfigurator* y debería funcionar todo sin mayor problema.

⁷Memoria virtual es la combinación de la RAM física y la cantidad de espacio swap que haya reservado a Linux.

⁸Conjunto de chips.

XFree86 tiene un archivo de configuración en `/etc/X11/XF86Config`. Este archivo está dispuesto en secciones con el siguiente formato:

```
Section "Nombre de la seccion"  
  Comando1 "Opción"  
  Comando2 "Opción"  
  Subsection "Nombre de la subsección"  
    Comando3 "Opción"  
  EndSubSection  
EndSection
```

Las secciones que podemos encontrar son:

- Modules
- Files
- ServerFlags
- InputDevice
- Monitor
- Device
- Screen
- ServerLayout

6.1.2. Descripción de Entornos de Escritorios

En esta parte, describiremos algunos entornos de escritorios, dentro de los cuales existen algunos que son más amigables que otros, como lo son Gnome y KDE. Otros por ejemplo, como Xfce o Black Box no poseen tanta facilidad, y su manejo se basa en consolas de texto.

6.1.2.1. Gnome

Es uno de los desktops más usados del mundo. Éste proporciona una amplia gama de proyectos, entre los cuales tenemos:

Gnome Desktop: Un escritorio de trabajo, basado en un ambiente de múltiples ventanas, que es fácil de utilizar por los usuarios.

Plataforma de desarrollo Gnome: Una rica colección de Herramientas, Bibliotecas, y componentes para desarrollar poderosas aplicaciones en UNIX/ Linux

Gnome Office: Un set de aplicaciones de oficina, compuesto por editores de texto (AbiWord), planillas de cálculos (gnumeric), browsers (galeon) y otros programas como sopidopi, gfax, gimp, Evolution, agnubis, MrProject y guppi entre otros.

6.1.2.2. KDE

Otro de los Desktops más usados en la actualidad y que por su fácil modo de operar, se ha vuelto uno de los favoritos de los usuarios. KDE, al igual que Gnome, proporciona un gran número de utilidades y aplicaciones como por ejemplo:

KDE Desktop Environment: Escritorio de trabajo completísimo, basado en ambientes de múltiples ventanas, que trae numerosas herramientas para poder manejar archivos, directorios, temas de escritorio, dispositivos, configuración del sistema y muchos otros.

KDE Application Framework Para desarrollar aplicaciones no sólo basadas en KDE, sino que también en Qt y así poder migrar a otras plataformas, los participantes de KParts, ayudan a los desarrolladores a crear y/o participar en nuevos proyectos.

6.1.2.3. Xfce

Xfce es un entorno de escritorio ligero para varios sistemas *NIX. Diseñado para la productividad, carga y ejecuta aplicaciones rápidamente, mientras conserva los recursos del sistema.

La última versión de Xfce (Xfce 4) posee una cantidad de características que pasaremos a revisar a continuación.

- Los estándares de Freedesktop permiten la interoperabilidad nativa con Gnome y KDE.
- A diferencia del XFce 3, el XFce 4 proporciona una estructura de desarrollo para aplicaciones.
- XFce4 proporciona el MCS, el Configurador Multi-Canal (Multi-Channel Settings), un sistema de administración modular, independiente del huésped, y transparente a la red. El MCS es compatible con los estándares de xsettings.
- Soporte para “Arrastrar y Soltar”.
- Fuentes Xft y soporte para fuentes anti-alias.
- Puede instalar/lanzarse cada módulo del XFce 4 por separado si no los necesita todos, o si tiene unos recursos de sistemas limitados.
- Administración de archivos con navegación por redes samba y la posibilidad de automontar/desmontar usando el contenido de fstab.
- Un panel principal, posicionable vertical u horizontalmente, de tamaño variable, con capacidad de auto-ocultado, con menús separables fácilmente configurables, además de lanzadores de aplicaciones.
- Un administrador de ventanas con más de 60 formas de decorarlo disponibles, soporte para Xinerama, sombreado de ventanas, cambio de escritorio virtual con la rueda del ratón, redimensionado de ventanas y cambio de posición desde atajos de teclado, etc. . .
- Un administrador de imágenes de fondos de escritorio, un menú de ventana raíz, y una utilidad para la configuración de márgenes.
- Un administrador centralizado.
- Una interfaz al sistema de impresión de documentos.
- Un paginador gráfico.
- Un motor de temas para Gtk2.
- Una barra de tareas.
- Traducción a 18 idiomas. . .

6.1.2.4. Otros

Cabe señalar que existen muchos otros entornos de escritorio, sólo por señalar algunos podemos nombrar a **Window Maker**, **Enlightenment**, **Black Box**, **Flush Box**, etc.. Todas las anteriores tienen un símil a Xfce, ya que no poseen un manejador de ventanas que facilite la utilización de los programas, sino que hay que hacerlo vía consola.

6.2. Aplicaciones de Oficinas

Muchas veces no se conocen aplicaciones para trabajar documentos cuya extensión es *.doc* o planillas (*.xls*). A continuación se darán a conocer ciertas aplicaciones que facilitan estas tareas y otras más.

6.2.1. Aplicaciones de KDE

El KDE Office Application suite trae consigo aplicaciones para uso en la oficina, como KWord, KSpread, KPresenter, KChart, KIllustrator.

6.2.2. OpenOffice

La misión de Open Office es crear, en el entorno de una comunidad, el suite de oficina internacional (líder) que trabajará en todas las plataformas principales y permitirá un acceso a toda la funcionalidad y datos por medio de APIs basados en componentes abiertos y un formato de archivos XML.

Es un suite de oficina de software libre⁹, gratis para todo el mundo, y que tiene módulos de procesador de textos, hoja de cálculo, gráficos vectoriales, edición HTML y presentaciones. Con respecto a su funcionalidad y su operatividad, compara favorablemente con todos los conocidos suites, y se debe considerar realmente como alternativa a aquellos. Dado que es un producto de libre difusión, es fácil evaluarlo en tu entorno particular. En todo caso, las versiones más recientes ya han demostrado su valor para muchas aplicaciones personales y profesionales.

6.2.3. Aplicaciones GNU en general

Existe una gran variedad de aplicaciones que a continuación nombraremos¹⁰. Por ejemplo, para ver archivos Post-Script está el programa *Gv*, que además sirve para ver Pdf's. Para estos últimos, también se encuentra el *Acroread* y *Xpdf*. También está el *Star Office*, que es similar a *Open Office*. Para las planillas de cálculo se encuentra también *Gnumeric*.

6.3. Herramientas de Mensajería, E-mail y Web

Parte importante hoy en día es la comunicación electrónica, y eso de ninguna manera se ha dejado de lado en *Linux*.

6.3.1. Mensajería

Hoy en día está muy de moda el protocolo *MSN*, pero sin duda no es el único. Es por eso que a continuación damos un vistazo a algunas aplicaciones de mensajería.

⁹Más información en www.openoffice.org.

¹⁰Obviamente sólo nombraremos algunas y no necesariamente las mejores. Eso siempre queda a criterio del usuario.

6.3.1.1. kopete(1)

Kopete es un cliente de mensajería instantánea diseñado a base de componentes (plugins). Kopete maneja los protocolos de mensajería como componentes que permiten una instalación, uso y configuración de una manera totalmente modular, sin saber la aplicación sobre los detalles de estos componentes. El objetivo de Kopete es proveer a los usuarios una interfaz homogénea entre todos los servicios de mensajería instantánea, y al mismo tiempo ofrecer a los desarrolladores una manera fácil de agregar nuevas funcionalidades o soportar nuevos protocolos. El equipo de Kopete ya ha desarrollado plugins para los protocolos más populares los cuales vienen listos para ser usados, y al mismo tiempo plantillas para que los nuevos desarrolladores puedan basar sus plugins. Kopete es un proyecto grande y ya soporta los protocolos Jabber, ICQ, AIM, MSN, Yahoo (en CVS), IRC, Windows LANs, GaduGadu, IRC y SMS.

Cabe señalar que el principal desarrollador, el autor original y líder del proyecto es el chileno Duncan Mac-Vicar Prett¹¹.

6.3.1.2. gaim(1)

Gaim es un clon del cliente de mensajería instantánea American Online, que usa GTK y toolkit. La aplicación ofrece más funcionalidad que el cliente oficial AIM, pero es más pequeño y rápido. Gaim trae múltiples características como opciones de grupo, opciones de Chat, funciones para la información del usuario y opciones de tiempo de las conversaciones.

6.3.1.3. xchat(1)

Xchat es un cliente IRC para sistemas operativos UNIX. Éste corre en la mayoría de los sistemas bajo BSD y POSIX, como los siguientes:

- Linux
- FreeBSD
- NetBSD
- OpenBSD
- Solaris
- AIX
- IRIX
- DEC/Compaq Tru64 UNIX
- HP-UX 10.20 y 11
- MacOS X
- Windows 9x/NT

Xchat es un cliente gráfico de IRC, y corre bajo el sistema X Windows usando GTK + toolkit. Opcionalmente puede ser compilado utilizando Gnome.

6.3.2. E-mail

Acá veremos algunas aplicaciones para recibir y enviar correos electrónicos.

¹¹Para mayor información, visita www.kopete.org.

6.3.2.1. kmail(1)

Kmail es un cliente de correos lleno de características que pertenece al entorno de escritorio KDE. Entre sus características destacan soporte para IMAP, POP3, múltiples cuentas, filtros poderosos, privacidad PGP/GnuPG. Para saber más sobre las características de Kmail, visite la siguiente dirección: <http://kmail.kde.org/features.html>.

6.3.2.2. mozilla(1)

Mozilla no sólo es un cliente de correos, sino que ofrece muchas cosas más. A continuación se listan algunas de las aplicaciones de Mozilla:

- Ofrece una suite llena de aplicaciones integradas de Internet incluyendo un buscador web, un cliente de E-mail, un libro de dirección, un compositor de Páginas Web, chat y calendario.
- Mozilla 1.5 es la suite más avanzada, la de más alcance de Internet:
 - Un buscador web moderno:
 - El buscador tabulado
 - Bloqueo pop-up
 - Controles avanzados de la privacidad y seguridad
 - Una barra con las opciones más recurrentes
 - Manejadores de contraseña
 - Los mejores estándares web y compatibilidad con el **wide** para la gran variedad que existe en Internet.
 - Software de E-mail dependiendo de la Empresa:
 - Sofisticado, adaptable, controlando el correo SPAM
 - Caja fuerte del virus que está afectando a otros clientes del E-mail
 - Cuentas múltiples del correo
 - Ver mensaje de correo nuevo
 - Comprobación del correo
 - Edición avanzada de la Página Web :
 - Edición del WYSIWYG de las páginas web
 - Una extensa lista con las características internacionales
 - Soporte uniforme a las plataformas (Windows, Mac de OS, Linux, y más)
- Liberar las descargas directas del software para Windows, Mac Os X, Linux, y más.
- Desde africano a Zulu, Mozilla se ha traducido a 50 idiomas desde el lanzamiento de Mozilla 1.0.

6.3.2.3. pine(1)

Pine®(un programa para noticias de Internet y correos electrónicos) es una herramienta para leer, enviar y manejar mensajes electrónicos. Pine fue desarrollado por Computing & Communications de la Universidad de Washington.

A pesar que Pine ha sido desarrollado para usuarios sin experiencia, ha evolucionado hasta el punto de dar soporte a características muy avanzadas, e incluso el número de configuraciones y opciones de preferencia personal han aumentado. Pine esta disponible para UNIX así como también lo está para computadores personales que están corriendo sistemas operativos Microsoft.

6.3.3. Web

En esta sección se darán a conocer algunos buscadores que nos servirán para navegar en la Red.

6.3.3.1. konqueror(1)

Konqueror es un manejador de archivos para el entorno de escritorio KDE¹². Éste da soporte a un manejador de archivos básicos dentro de un sistema de archivos UNIXy con un simple copiar-pegar se pueden realizar operaciones remotas y de una red local.

Konqueror es el soporte para las últimas tecnologías de KDE, además es *open source* con HTML4.0, dando soporte para Java applets, Java Script, CSS1 y (parcialmente) CSS2, como también plug-in de *netscape(1)*, como Flash o RealVideo.

Finalmente podemos decir que Konqueror es una aplicación mundialmente conocida. Entre sus características se encuentra una muy buena representación de HTML, llamada khtml. Ésta está implementada como un Kparts y como tal, puede ser fácilmente usada por otros programas de KDE.

Por último podemos nombrar que tiene un gran soporte para caracteres árabes y hebreos, además de dar soporte a SSL, para lo cual requiere Open SSL.

6.3.3.2. opera(1)

Opera es buscador pequeño amigable, seguro y muy rápido, incluso con todas sus aplicaciones instaladas y en un sistema con recursos limitados. Algunas de sus características son:

- Manejador de Password, para mayor seguridad.
- Un poderoso panel para manejarse.
- Una agenda para anotar cosas importantes y envío de mensajes electrónicos.
- Botones que utilizan un atajo para realizar operaciones, lo que hace más rápido su uso.
- Manejador de Cookies.
- Un cliente de correos que soporta POP3. IMAP y ESMTP...

6.3.3.3. lynx(1)

A diferencia de los anteriores, Lynx es un buscador que no posee entorno gráfico, por lo que todas las operaciones se realizan en consolas de texto. Para ejecutarlo debes escribir su nombre en una consola y este comenzará a ejecutarse. Por lo mismo, en este caso hay que olvidarse del ratón, y sólo trabajar moviendo las flechas y apretando tab.

Cabe recordar que su uso si bien no es amigable, posee grandes potenciales y es sumamente rápido. Más bien lo nombramos con el objetivo de que se sepa que así se empezó a navegar en Internet, y que en un comienzo no era todo tan bonito. Si te interesa saber más, visita el sitio web www.lynx.org.

6.4. Juegos

Una de las grandes cosas por lo que mucha gente no se interesa en conocer Linux es por que se dice que no puedes jugar, lo que por supuesto no tiene nada de cierto. A continuación se darán a conocer algunos juegos pertenecientes a los distintos entornos de escritorios.

¹²No obstante éste puede ser visto desde cualquier entorno de escritorio. Basta con ejecutarlo desde una consola.

6.4.1. Gnome Games

Gnome trae una gama enorme de aplicaciones para entretenerse en los ratos de ocio. Entre estos podemos nombrar algunos como FreeCell, Gnibbles, gataxx, Glines, Gnome-Stones, Gnobotsll, Gnotravex, etc.. Entre éstos se encuentran juegos de estrategia, juegos de mesa entre otros.

6.4.2. KDE Games

Al igual que Gnome, KDE posee una gran cantidad de juegos arcade, estrategia, lógica, entre otros. Por nombrar algunos, podemos citar a Kasteroids, Kbounce, Kolf, KBattleship, Kwin4, Shisen-Sho, Kpoker, Katomic entre otros.

6.4.3. Tux Racer

Este ha sido el estandarte de los juegos Open Source. Se trata de un pingüino (tux), que debe deslizarse por el Hielo, agarrando pescados en su camino. Es muy parecido a los clásicos “Juegos de Invierno”, y existe una versión completamente Open Source.

6.4.4. Juegos Gratuitos

Generalmente, cuando se liberan juegos gratuitos, crean una versión para Linux. Esto, ya que técnicamente es bastante sencillo portar un juego para Linux, si se tiene el código de fuente. Entre estos, encontramos Army Ops o Enemy Territory, ambos de primera persona. El primero es uno de los juegos más reales que existen hoy en día, creado por “U.S. Army” para incentivar a los Americanos a incorporarse a sus filas. El segundo, esta ambientado en la Segunda Guerra Mundial, y es uno de los juegos más populares de nuestros días.

6.4.5. Juegos Comerciales

Existen varios juegos comerciales, que liberan versiones para Linux. Estos, generalmente son aquellos juegos que utilizan OpenGL como motor gráfico. Entre estos encontramos Quake, Quake2, Quake3, RTCW, UT, UT2000, etc.