

Lenguajes de programación

***Arquitectura y Sistemas Operativos
Tecnicatura Superior en Programación.
UTN-FRA***

Autores: Prof. Martín Isusi Seff

Revisores: Prof. Marcos Pablo Russo

Versión : 1



Esta obra está bajo una [Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional](#).

Lenguajes de programación.

Para poder comenzar a hablar de los lenguajes de programación, sus características, los distintos tipos, etc., es necesario responder una pregunta fundamental: ¿Qué es un lenguaje de programación?

Un lenguaje de programación es un lenguaje especial, que especifica su propia sintaxis, reglas y un conjunto de palabras clave, utilizado para poder comunicar instrucciones a una máquina, más específicamente, a una computadora. Los lenguajes de programación, son los que nos permiten el desarrollo del software.

Durante la primera generación de computadoras (1938-1952) y hasta la creación de las primeras tarjetas perforadas, la lógica que ejecutaba cualquier sistema de computación se almacenaba de manera cableada en el equipo. La implementación de las tarjetas perforadas como medio de almacenamiento a principio de los años '50 (en el final de la 1ra generación de computadoras), da lugar a lo que se conoce como la primera generación de lenguajes de programación. En esta primera generación, los programas eran escritos directamente en lenguaje de máquina. No existía abstracción alguna, por lo que las instrucciones que escribía el programador se ejecutaban directamente sobre el hardware del equipo.

Coincidiendo con el inicio de la segunda generación de computadoras (1953-1962), surgen los primeros lenguajes de ensamblado. Esto da lugar a una segunda generación de lenguajes de programación.

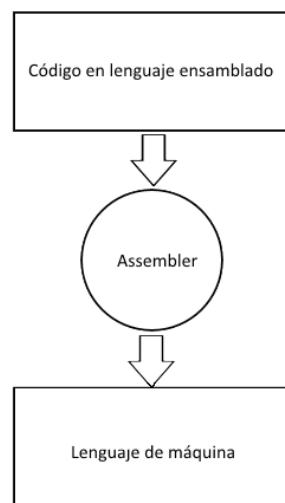
Por último, a finales de los años '50, el desarrollo de nuevos lenguajes de programación como C, COBOL, BASIC, y muchos de los lenguajes de programación actuales, corresponden a la tercera generación de lenguajes de programación. Su meta, fue abstraer al programador del hardware del sistema.

Niveles de abstracción en los lenguajes de programación

Los lenguajes de programación suelen clasificarse por su nivel de abstracción con respecto al hardware, por lo tanto, tendremos los lenguajes de programación de bajo nivel y los lenguajes de programación de alto nivel.

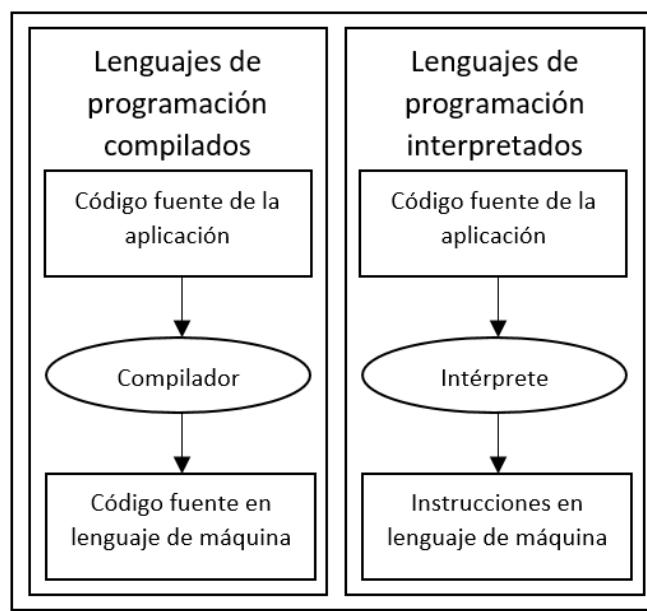
Dentro de los lenguajes de programación de bajo nivel tenemos al lenguaje de máquina y los lenguajes de ensamblado. El lenguaje de máquina es el lenguaje con el que trabaja el hardware y es, de hecho, el único con el que puede trabajar. Para el ser humano este lenguaje no es más que una secuencia de dígitos binarios, por lo que no es utilizado directamente por los programadores.

Siendo el lenguaje de máquina el código con el que trabaja el hardware directamente, podemos decir que no posee abstracción alguna, por lo tanto, el primer nivel de abstracción corresponde a los lenguajes de ensamblado. Estos lenguajes consisten de una serie de instrucciones que se ejecutan directamente sobre el microprocesador y sus componentes (registros, contadores, etc.). Generalmente, estas instrucciones se representan con una abreviación o palabra clave tales como MOV, JUMP, ADDL, siendo cada una, una instrucción distinta en el microprocesador. Dado que el hardware sólo puede ejecutar instrucciones almacenadas en lenguaje de máquina, debe existir un componente encargado de traducir las instrucciones en lenguaje de ensamblado a lenguaje de máquina. Este componente es el ensamblador o assembler.



Por encima de los lenguajes de programación de bajo nivel, nos encontramos con los lenguajes de programación de alto nivel. Estos son lenguajes de programación que tienen como característica principal, la abstracción del hardware del equipo. Para poder escribir un programa en un lenguaje de programación de alto nivel, no es necesario conocer la arquitectura del sistema, algo que sí es necesario cuando trabajamos, por ejemplo, con un lenguaje de ensamblado. Dentro de los lenguajes de programación de alto nivel, también podemos identificar distintos niveles de abstracción. Esto quiere decir que algunos lenguajes de programación de alto nivel nos proveen herramientas para manejar aspectos del hardware directamente, como, por ejemplo, el manejo de punteros en C. Revisando el ejemplo anterior, podemos decir que C es un lenguaje de programación de alto nivel, pero dentro de este grupo se encuentra entre los que tienen el nivel de abstracción más bajo. Un ejemplo contrario podría Javascript, cuya sintaxis y funcionalidad se encuentra muy abstracta del hardware, por lo que no podemos utilizarlo para realizar desarrollos que controlen el hardware directamente.

Lenguajes de programación de alto nivel



Distintos tipos de lenguajes de programación: Interpretados y compilados

Debido a que el sistema sólo es capaz de ejecutar las instrucciones almacenadas en lenguajes de máquina, los lenguajes de programación de alto nivel necesitarán de un componente que "traduzca" las instrucciones escritas por el programador, al lenguaje de máquina. Este componente puede ser un compilador o un intérprete. Esto hace que dentro de los lenguajes de programación de alto nivel existan dos grandes tipos: Lenguajes de programación compilados y lenguajes de programación interpretados.

Los lenguajes de programación compilados son aquellos que se traducen enteramente al lenguaje de máquina para ser ejecutados directamente en el hardware, mientras que los lenguajes de programación interpretados son aquellos en los que necesitan de un software intermedio llamado intérprete que será el encargado de ejecutar las instrucciones. Generalmente, el intérprete sí está almacenado en lenguaje de máquina.

Lenguajes de programación compilados

Los lenguajes de programación compilados son aquellos que utilizan un compilador para transformar el código fuente¹ de la aplicación en código de máquina que pueda interpretar el sistema. El compilador tomará los archivos que contienen el código fuente y dará como

¹ Cuando hablamos de código fuente, nos referimos al código escrito por el programador, en un determinado lenguaje de programación y que se almacena en archivos cuya extensión dependerá del lenguaje de programación utilizado. Por ejemplo, el código fuente de una aplicación desarrollada en lenguaje C, corresponderá a los archivos con extensión .c que correspondan a la misma.

resultado, archivos con el código de máquina.

interpretar el sistema. El compilador tomará los archivos que contienen el código fuente y dará como resultado, archivos con el código de máquina.

La principal ventaja de los lenguajes compilados sobre los interpretados, es la velocidad de ejecución. Dado que el compilador traduce el código fuente y lo almacena en código de máquina, la ejecución de estos archivos resulta mucho más rápida y eficiente que la ejecución de instrucciones en un lenguaje interpretado.

Dentro de los lenguajes de programación de alto nivel, aquellos que tienen a tener un menor nivel de abstracción suelen ser compilados ya que, al almacenarle finalmente en lenguaje de máquina, permiten ejecutar instrucciones directamente sobre el hardware. Algunos ejemplos de estos lenguajes son C, C++, etc.

Entre las desventajas que podemos mencionar de los lenguajes compilados, se encuentra la necesidad de compilar el código fuente antes de poder ejecutar la aplicación. Si bien una vez compilada, la ejecución es más rápida, el hecho de que haya que compilar siempre antes de ejecutar hace que los tiempos de desarrollo sean más lentos. Otra desventaja radica en el hecho de que, como cada arquitectura de hardware posee su propio lenguaje de máquina, las aplicaciones escritas en lenguajes compilados deben ser compiladas para cada arquitectura que se deseé, por lo tanto, podemos decir que los lenguajes de programación compilados son dependientes de la plataforma en la que se ejecutan.

Algunos ejemplos de lenguajes de programación compilados son C, C++, C#, Java², Basic, etc.

Lenguajes de programación interpretados

En contraste con los lenguajes de programación compilados, los lenguajes de programación interpretados no necesitan ser traducidos a código de máquina para ejecutarse. O al menos, no directamente. En el caso anterior, el compilador traducía el código fuente completo y lo almacenaba en lenguaje de máquina. Aquí, tendremos un intérprete, que será el encargado de traducir el código fuente a código de máquina, pero en lugar de almacenarlo en un archivo, ejecutará las instrucciones directamente. Para poder hacer esto, el intérprete lee línea por línea cada instrucción del código fuente, y las ejecuta independientemente de manera

² A pesar de que Java es un lenguaje de programación compilado, una vez realizada la compilación del código fuente, los archivos resultantes no contienen código de máquina. El código almacenado en los archivos resultantes de la compilación, contienen un código de bytes o *bytecode* que será ejecutado por una “máquina virtual” Java. Será luego la máquina virtual Java la encargada de traducir ese bytecode a código de máquina nativo.

ordenada.

La principal desventaja de este tipo de lenguajes es la velocidad de ejecución. Dado que el código nunca se encuentra almacenado en lenguaje de máquina, cada ejecución requiere de la “traducción” de las instrucciones. De todas maneras, al no requerir una compilación cada vez que se ejecutan las aplicaciones, el desarrollo de las mismas se realiza de una manera más ágil.

Algunos ejemplos de lenguajes de programación interpretados son PHP, Perl, Ruby, JavaScript, Python, etc.

Clasificación de los lenguajes de programación dependiendo del paradigma

Dentro del ámbito del desarrollo de software, podemos encontrar distintos paradigmas o “formas” de programación. Existen paradigmas orientados a la metodología de trabajo, que imponen reglas o formas de trabajar desde lo práctico. Por otro lado, podemos identificar paradigmas de programación que dependen del lenguaje. Algunos ejemplos de paradigmas son la programación orientada a objetos, programación procedural, programación funcional, etc. Las características de cada lenguaje de programación serán las que ubiquen al mismo dentro de alguno de los paradigmas.

Resumen

Los lenguajes de programación son la forma en la que el programador puede dar las instrucciones a la computadora para que la misma realice distintas operaciones. Los lenguajes de programación pueden clasificarse por su nivel de abstracción, encontrando en el nivel más bajo el lenguaje de máquina y los lenguajes de ensamblado. Sobre los lenguajes de programación de bajo nivel, en un nivel superior, encontramos los lenguajes de programación de alto nivel. Estos lenguajes son aquellos que buscan abstraer al programador del hardware, es decir que la persona no necesite desarrollar para distintos dispositivos la misma aplicación. Dentro de los lenguajes de programación de alto nivel podemos identificar dos grandes grupos: Lenguajes de programación compilados y lenguajes de programación interpretados. Por último, dentro de la programación existen distintos paradigmas, que corresponden a distintas “formas” de programar. Las características de cada lenguaje de programación son las que ubican a cada uno en alguno de los paradigmas.

Elementos fundamentales de los sistemas operativos

*Arquitectura y Sistemas Operativos
Tecnicatura Superior en Programación.
UTN-FRA*

Autores: Prof. Martín Isusi Seff

Revisores: Prof. Marcos Pablo Russo

Versión : 1



Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

El sistema operativo, desde el punto de vista del software, es aquel que provee las funciones fundamentales a una computadora. Es aquel que permite interactuar con el hardware, define la interfaz con el usuario y da las herramientas básicas para que una computadora sea útil.

¿Qué es el kernel?

El kernel, o núcleo del sistema operativo es el componente responsable de realizar el manejo de las funciones de bajo nivel de la computadora, por ejemplo:

- Ubicar un programa en la memoria
- Asignar el tiempo de CPU de un programa
- Hacer de interfaz entre el software y los dispositivos
- Permitir la interacción entre diferentes programas

Cuando uno usa cualquier programa, este recae en el kernel y muchas de sus funciones básicas. Por ejemplo, un navegador de internet se comunica con el exterior utilizando las funciones de red del kernel. Además, será el kernel quién ubique al navegador en la memoria principal y asigne el tiempo de la CPU sin el cual el navegador no podría funcionar. Si el navegador necesita de algún software adicional para mostrar, por ejemplo, contenido multimedia, será el kernel quién administre la comunicación entre ambos programas.

Podríamos decir que el kernel es el componente que mantiene unidas todas las partes de una computadora, sin él, las computadoras serían prácticamente inútiles.

Cada sistema operativo tiene su propio kernel, es decir que, Mac OS X tiene un kernel distinto al que podemos encontrar en cualquier versión de Windows o las distribuciones de Linux. Cada uno está diseñado de una manera distinta, por lo que provee las funcionalidades de una manera diferente, aunque la funcionalidad general sea la misma.

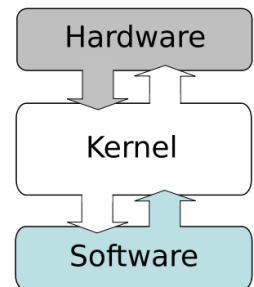


Figura 1. Esquema de interacción entre el hardware, el kernel y el software

Además del kernel, ¿Qué diferencia un SO de otro?

El kernel es el componente central de cualquier sistema operativo, pero no es algo que el usuario manipule directamente. En su lugar, la mayoría de los usuarios interactúa con distintos componentes de software, muchos de los cuales están asociados con un sistema operativo en particular. Estos programas incluyen:

- **Intérprete de línea de comandos o command-line shells.** Hace algunos años, los usuarios interactuaban con la computadora exclusivamente escribiendo comandos en un

programa conocido como shell o intérprete de línea de comandos. Tales comandos se utilizaban para renombrar archivos, ejecutar programas o realizar cualquiera de las funciones que permitía el sistema operativo. Aunque hoy en día la mayoría de los usuarios no utiliza (y en algunos casos ni conoce) la línea de comandos, esta interfaz sigue siendo imprescindible en muchos casos. Existen distintos tipos de shells, entre los cuales se encuentran sh (bourne shell), bash (bourne-again shell), csh (c shell), entre otros.

- **Interfaz gráfica.** La interfaz gráfica o GUI (Graphical User Interface) es una interfaz que proporciona un uso mucho más intuitivo de la computadora. En lugar de ejecutar comandos, una interfaz gráfica basa su manejo en íconos, menús, y punteros de mouse. Microsoft Windows y Mac OS tiene sus propias interfaces gráficas, mientras que Linux se basa en una interfaz gráfica conocida como X Windows System o simplemente X. X es una interfaz de usuario básica, por lo que las distintas distribuciones basadas en Linux utilizan además paquetes de software conocidos como entornos de escritorio. Algunos ejemplos de entornos de escritorio son GNOME, KDE, XFCE, etc.
- **Programas utilitarios.** Hoy en día, la mayoría de los sistemas operativos se instalan con varios programas utilitarios como calculadoras, calendarios, aplicaciones para el mantenimiento de discos, etc. Estos programas varían entre los distintos sistemas operativos, así como el nombre, y la forma de usarlos de cada uno¹.
- **Librerías.** Las librerías son archivos que contienen las funciones básicas que utilizan las distintas aplicaciones. Existen librerías específicas para cada aplicación o para cada funcionalidad del sistema operativo. Cada sistema operativo tiene su propio conjunto de librerías y, además, puede tener librerías para distintas arquitecturas de hardware, por ejemplo, las distribuciones basadas en GNU/Linux tienen todas las librerías ubicadas en un directorio llamado lib, pero, además, existe lib64 que almacena las mismas librerías, pero compiladas para la arquitectura de microprocesadores de 64 bits. Cuando un sistema contiene tanto los directorios lib y lib64 se dice que el sistema operativo es Multilib.
- **Programas varios.** Cada sistema operativo tiene su propio conjunto de aplicaciones de uso común como navegadores, software de ofimática, etc. Aunque generalmente estas aplicaciones no son un componente del sistema operativo, estrictamente hablando, pueden estar instaladas por defecto en ellos.

Uso básico de las distintas interfaces de usuario

Uso de la interfaz de línea de comandos

¹ Existen páginas en las que podemos encontrar las equivalencias entre aplicaciones para los distintos sistemas operativos. Algunos ejemplos son www.linuxrsp.ru/win-lin-soft/table-eng o www.linuxalt.com.

Hoy en día, la mayor parte de los usuarios utiliza una interfaz gráfica, independientemente del sistema operativo. Usamos una interfaz gráfica en la computadora, en un dispositivo móvil o en una tele. Esto no quiere decir que la interfaz de línea de comandos no sea útil, de hecho, es todo lo contrario. La interfaz gráfica es muy importante para el uso general de una computadora. Realizar tareas

básicas es siempre más fácil con la ayuda de íconos, un cursor, o cualquiera de los elementos que proveen las interfaces gráficas. Pero algunas tareas que no son de uso general, no son tan simples o no es tan conveniente realizarlas con una interfaz gráfica y, por el contrario, es mejor usar una línea de comandos. Un ejemplo podría ser la creación de usuarios en un sistema: Si tengo que crear mil usuarios en un sistema operativo, con una interfaz gráfica puede ser una tarea tediosa, porque implica que deba moverme por los diferentes controles, ingresando la información, aceptando mensajes de alerta, etc. Automatizar una interfaz gráfica no es una tarea simple y siempre requiere de algún software externo. Por el contrario, si esta tarea fuera hecha a través de una interfaz de línea de comandos, probablemente la creación de los usuarios se realice con un solo comando lo cual simplificaría el asunto y, además, las interfaces de línea de comandos son muy simples de automatizar. Los servidores no utilizan interfaz gráfica y su administración es a través de línea de comandos.

Uso de la interfaz gráfica

La mayoría de los usuarios prefiere utilizar interfaces gráficas (GUI por sus siglas en inglés. Graphical User Interface), dada la simplicidad y la manera intuitiva en la que presenta la información. Es por eso que la mayoría de los sistemas operativos orientados al usuario común se instalan con una interfaz de usuario.

A diferencia de Windows y OS X, GNU/Linux provee distintas interfaces gráficas, o, entornos de escritorio. Los entornos de escritorio dependen de la distribución de GNU/Linux que se esté utilizando y de las aplicaciones que se elijan al momento de la instalación. Las opciones más comunes en lo que respecta a entornos de escritorio en Linux son: GNOME, KDE, XFCE, y Unity (Basado en GNOME y que actualmente es el escritorio por defecto de Ubuntu).

¿Qué rol cumple GNU/Linux dentro de los SO?

Si bien el apunte estará enfocado en el análisis de la arquitectura y los componentes que conforman un sistema operativo, la mayoría de los análisis prácticos estarán hechos

```

GNU nano 2.2.4          FILE: /mnt/etc/fstab

# /etc/fstab: static file system information
#
# <file system>      <dir>        <type>     <options>      <dump> <pass>
devpts                 /dev/pts      devpts      defaults        0   0
shm                    /dev/shm      tmpfs      nodev,nosuid    0   0
#/dev/cdrom             /media/cd     auto       ro,user,noauto,unhide 0   0
#/dev/dvd               /media/dvd   auto       ro,user,noauto,unhide 0   0
#/dev/fd0               /media/f1     auto       user,noauto      0   0

UUID=6dad4fac-0f3c-4939-80d6-f2004d47c166 / ext3 defaults 0 1
UUID=74650592-b40e-448b-a2df-82867c756f04 swap swap defaults 0 0
UUID=c69098b4-a0d5-451c-9935-dc2ad4a29949 /home ext3 defaults 0 1
UUID=d38a7644-c989-4d46-95f9-b44998d9ffef /boot ext2 defaults 0 1

[ Read 15 lines ]
[G Get Help] [O WriteOut] [R Read File] [Y Prev Page] [K Cut Text] [C Cur Pos]
[X Exit] [J Justify] [W Where Is] [U Next Page] [U Uncut Text] [T To Spell]

```

Figura 2. El editor de texto GNU nano, incluido en la mayoría de los sistemas operativos GNU/Linux, es un ejemplo de aplicación que se ejecuta en línea de comandos en lugar de tener una interfaz gráfica.

puntualmente sobre sistemas operativos basados en GNU/Linux. Por lo tanto, será necesario conocer las diferencias entre los distintos tipos de sistemas operativos.

Comparando GNU/Linux con Unix

Unix es un sistema operativo creado en 1969 en los laboratorios Bell de la compañía AT&T. Inicialmente, este SO estaba escrito en un lenguaje de ensamblado, lo que hacía que sólo pueda ser ejecutado en configuraciones de hardware muy específicas. Es por eso que, en el año 1972, el sistema operativo completo se reescribió en lenguaje C y de ahí en más, fue posible la compilación para distintos dispositivos. Esto hizo que sea posible la revisión y recompilación del código fuente por parte de distintos equipos de desarrollo, tal es el caso del equipo de investigación y desarrollo de la Universidad de Berkeley que en 1977 desarrolla BSD, que era un sistema operativo completo basado en la sexta versión de Unix. Ante el avance de BSD, Bell realiza una demanda contra la Universidad de Berkeley, argumentando que BSD contenía código que era propiedad de la empresa. Luego de esta demanda, el desarrollo de sistemas operativos basados en Unix se vio limitado, es por eso que, en 1983, Richard Stallman decide crear GNU que comprenderá una copia de todas las aplicaciones que conformaban Unix, sin utilizar el código fuente del mismo. Para poder completar un sistema operativo completo, al proyecto GNU le faltaba un kernel. Esto se solucionó en el año 1991 con el desarrollo del kernel Linux por Linus Torvalds. El conjunto de GNU y Linux da como resultado la mayoría de los sistemas operativos actuales, que nosotros llamamos simplemente "Linux" aunque en realidad deberían ser llamados GNU/Linux.

Para concluir la comparación, podemos decir que GNU/Linux (muchas veces se dice Linux y esto en realidad está mal dicho, ya que Linux es el kernel y GNU las aplicaciones) puede ser considerado con un miembro de la familia de sistemas operativos Unix, aunque técnicamente no lo sea. Debido a su popularidad, GNU/Linux es compatible con una variedad más amplia de hardware que los sistemas operativos íntegramente basados en Unix.

Comparando GNU/Linux con Mac OS X

Hoy en día, el sistema operativo más utilizado en dispositivos de escritorio o notebooks es Windows, pero, ¿Cuál es la diferencia entre Windows y GNU/Linux?

La mayor diferencia radica en la licencia. Los sistemas operativos GNU/Linux son de código abierto, mientras que Windows se distribuye en base a una licencia comercial de software propietario. Por otro lado,

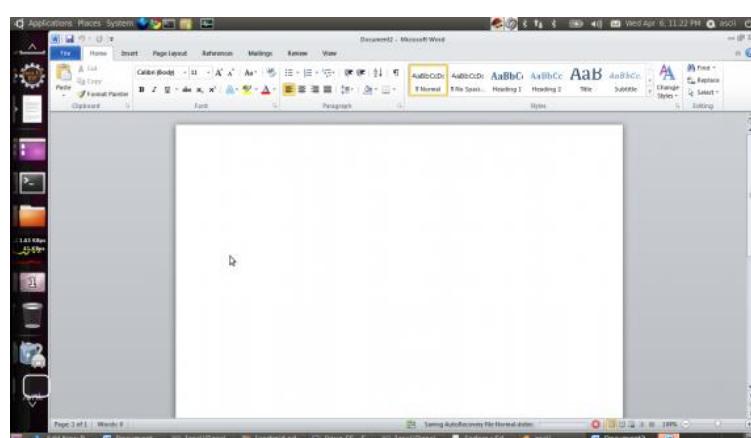


Figura 3. Microsoft Office Word, ejecutándose en Ubuntu, utilizando WineHQ

muchas distribuciones de Linux son gratuitas (al menos para su instalación), esto hace que el costo sea otra de las grandes diferencias, ya que todas las versiones de Windows se comercializan.

Con respecto a la compatibilidad, generalmente los fabricantes de hardware desarrollan los drivers específicamente para ser instalados en las diferentes versiones de Windows, aunque muchas veces también proveen de los drivers necesarios en su versión para GNU/Linux. Muchas veces, ciertos componentes suelen funcionar mejor bajo Windows, ya que fueron fabricados específicamente para ese sistema operativo, tal es el caso de las placas de video, que generalmente suelen funcionar mejor en las distintas versiones de Windows. Pero este no es siempre el caso, ya que Windows es un sistema operativo comercial, esto hace que cada vez que una nueva versión del sistema operativo sale al mercado, los desarrollos se orienten al mismo, dejando atrás los desarrollos para versiones anteriores. Esto aplica también a los drivers, por lo tanto, muchas veces no es posible utilizar ciertos dispositivos "viejos" en nuevas versiones de Windows. Linux, por su parte, recibe soporte de los fabricantes, pero también existen muchos drivers alternativos que son de código abierto. Generalmente estos drivers se mantienen a pesar del desarrollo de nuevas versiones del sistema operativo, por lo que es posible utilizar hardware "viejo" con nuevas versiones del SO.

Una de las principales comparaciones que muchas veces deja afuera la opción de un sistema operativo GNU/Linux es la disponibilidad de aplicaciones. Al ser Windows el sistema operativo más instalado en equipos de escritorio y notebooks, la mayoría de las aplicaciones suele desarrollarse para esta plataforma. Tal es el caso de Office, Photoshop, etc. De todas maneras, existen alternativas a cada una de estas aplicaciones, disponibles para las distribuciones de GNU/Linux. Tal es el caso que, por cada aplicación o suite de aplicaciones para Windows, existen una o más de una alternativa en GNU/Linux. Existe, además, la posibilidad de instalar librerías que hacen posible la ejecución de aplicaciones Windows en Linux. La librería más conocida corresponde al software WineHQ, que hace posible la ejecución de la mayoría de las aplicaciones de Windows bajo el entorno GNU/Linux.

En cuanto a la interfaz gráfica, al igual que OS X, Windows posee su propia interfaz de usuario. Desde las primeras versiones de Windows, lo que se intentó hacer fue crear un sistema operativo diseñado para el usuario común, por lo tanto, la interfaz gráfica siempre fue amigable e intuitiva. Esto hizo que hoy en día Windows sea el SO más popular. De todas maneras, los entornos de escritorio disponibles para GNU/Linux fueron evolucionando y hoy pueden competir sin dudas con la interfaz de Windows. Existen, de hecho, muchos entornos de escritorio que plantean una similitud con la interfaz de Windows en cuanto al diseño (XFCE, LXDE, etc).

Como último punto de comparación tenemos la configuración y seguridad. Con configuración, nos referimos a la posibilidad de modificar el SO y adaptarlo a cada necesidad particular. En

este punto, GNU/Linux es superior a Windows. Evidencia de esto es la innumerable cantidad de sistemas operativos basados en GNU/Linux y orientadas a distintos usos, por ejemplo, Scientific Linux, orientado a grandes aplicaciones científicas (utilizado por la Organización Europea para la Investigación Nuclear (CERN) en el acelerador de partículas más grande del mundo), Ubuntu, orientado al usuario medio, Kali Linux, orientado a seguridad informática, Huemul, orientado a informática forense etc. Si hablamos de seguridad, la arquitectura bajo la cual se desarrolla Linux, provee un uso más seguro del sistema operativo, limitando a los usuarios de una manera más estricta en la que se maneja Windows.

Para resumir, por casi dos décadas, Windows fue el sistema operativo dominante en lo que respecta a computadoras de escritorio y notebooks. Tanto en uso particular como en uso de oficina, los usuarios fueron familiarizándose con la interfaz de ese sistema operativo y muchas de las aplicaciones de uso general como Microsoft Office, etc. Si bien GNU/Linux podría ser utilizado en esos casos, el hecho de que los usuarios estén acostumbrados a utilizar Windows hace que el cambio hacia GNU/Linux sea una ardua tarea. Unix y GNU/Linux particularmente, han sido dominantes en el mercado de los servidores, debido a su manejo de la seguridad y los permisos, el rendimiento de los mismos, y las posibilidades de configuración y escalabilidad. Si bien Windows comercializa una versión dedicada a ser utilizada como servidor, nunca logró captar la mirada del mercado.

¿Qué es una distribución?

Hasta ahora, esta palabra se presentó varias veces en el texto, pero ¿Qué es una distribución específicamente? ¿A qué nos referimos con 'distribución'?

Nos referimos a distribución al conjunto del kernel de Linux, el conjunto de aplicaciones fundamentales que hacen al sistema operativo, y las configuraciones del mismo.

Desde el comienzo, numerosas distribuciones de GNU/Linux se fueron desarrollando, como se muestra en el siguiente gráfico:

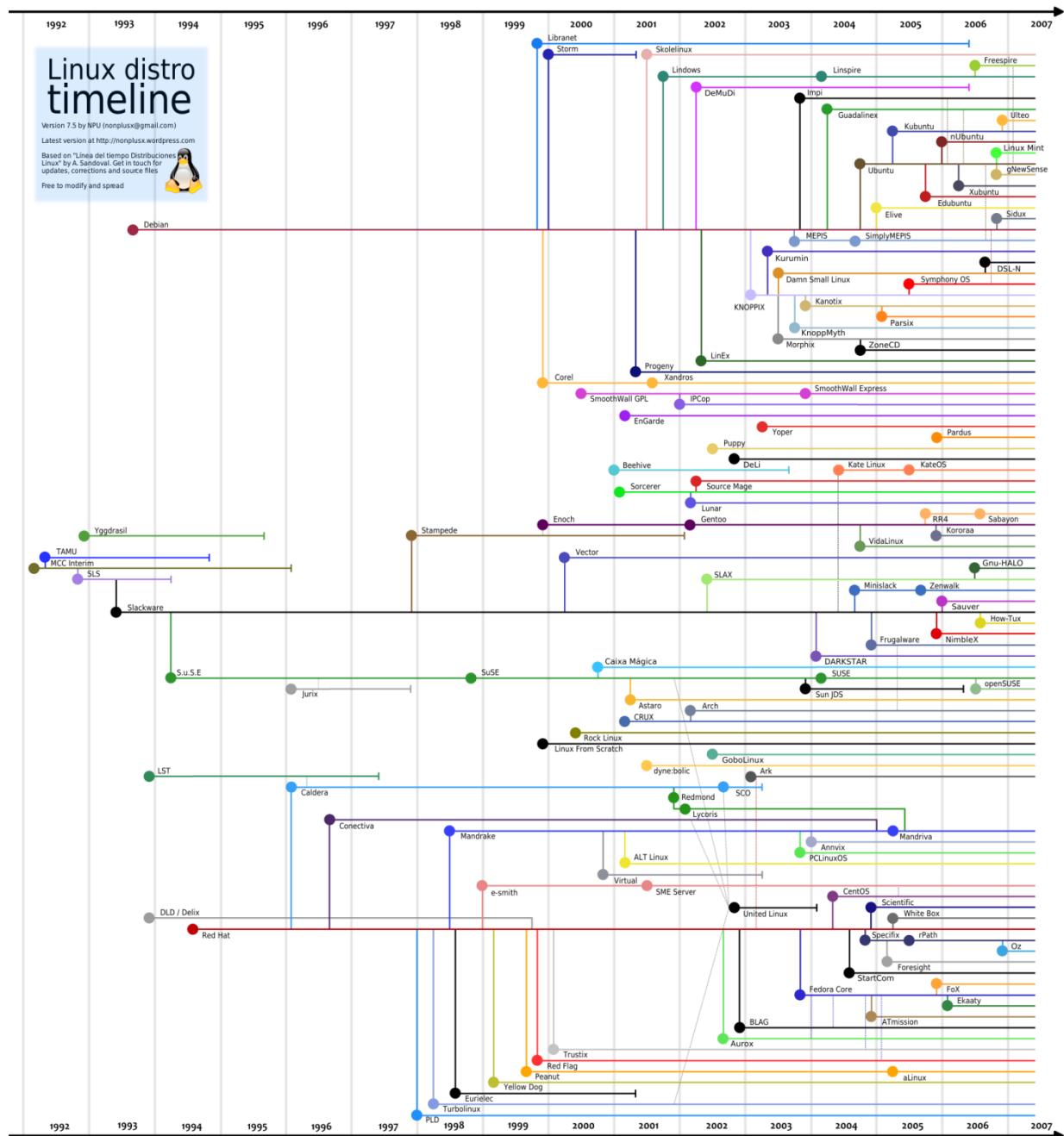


Fig. 1 Si bien el gráfico muestra las distribuciones, la intención del mismo es demostrar la enorme cantidad de distribuciones surgieron, y dar una idea de la cantidad de distribuciones que aún están por surgir.

Entendiendo los distintos tipos de licencia de software

***Arquitectura y Sistemas Operativos.
Tecnicatura Superior en Programación.
UTN-FRA***

Autores: Prof. Martín Isusi Seff

Revisores: Prof. Marcos Pablo Russo

Versión : 1



Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

Durante los años '50 y '60, con el advenimiento de los lenguajes de programación de alto nivel, la industria del software comenzó a crecer. La mayoría de los desarrollos eran producidos por academias y centros de investigación que trabajaban de manera conjunta y los publicaban como software de dominio público. La publicación de los desarrollos se realizaba en revistas o libros, y el contenido no era más que el código fuente. Cada persona que tuviera acceso a estas publicaciones, podía estudiar el código, modificarlo, mejorarlo o adaptarlo a las necesidades. A pesar de que en estas épocas el software se distribuía de manera libre, esto se daba por muchos factores. Uno de los factores radicaba en el hecho de que, para que un desarrollo sea compatible con distintos tipos de hardware, cada equipo debía realizar sus modificaciones. Otro de los factores era la seguridad. Entre los grupos de desarrollo existía (y sigue existiendo con justa razón) la idea de que cualquier desarrollo que no se publique con el código fuente podía contener una puerta trasera o backdoor que permita un ataque al sistema. Cabe mencionar, que en ese momento no existían los mecanismos de seguridad como los conocemos ahora. Además, los sistemas operativos utilizados en ese momento, no tenían un mecanismo de log, que lleve un registro de qué acciones realiza, por lo que cualquier software podía estar realizando cualquier acción sin que el usuario esté al tanto. Tal es el punto que, en algunos equipos de trabajo se manejaba la política de que todo software instalado debía tener su código libre, así se podía estar seguro de cuáles son las instrucciones que realizaba.

A finales de los años '60 y principios de los '70, con la aparición y la evolución de los sistemas operativos y los lenguajes de alto nivel, los precios del software empezaron a subir y los fabricantes de hardware empiezan a comercializar sus productos que incluían un paquete de software especialmente preparado para funcionar en ese equipo. Esto hace que el precio del software esté incluido en el precio del equipo, y sea comparativamente más económico frente al software que se comercializaba sin el hardware. Es por estos motivos que a fines de los '70 muchas empresas dedicadas al desarrollo de software, empiezan a limitar la distribución de sus aplicaciones y el código fuente de las mismas. Estas limitaciones, estaban abaladas por organismos legales, por lo tanto, esta formalización de la industria del software termina transformando de la misma un gran negocio. A pesar de las penalizaciones y multas que se aplicaban a aquellos que no se acaten a las leyes sobre distribución de software.

Hoy en día, a más de 50 años de los primeros pasos de comercialización de software, la distribución ilegal de software sigue estando muy presente y existe una lucha constante entre los desarrolladores del software privativo (aquel que no tiene su código libre) y los defensores de la libre distribución del software. Con el inmenso crecimiento de las redes, el control sobre el contenido que se distribuye se transforma en una tarea compleja, y muchas veces la falta de leyes hace que gran parte del software privativo pueda encontrarse ilegalmente publicada.

¿Qué son las licencias de software? Distintos tipos

El software es un tipo de propiedad intelectual que está gobernado por las leyes de copyright y en algunos países, se encuentran patentados. En general, esto quiere decir que es ilegal realizar copias de un software determinado, a menos que seas el desarrollador o tengas permiso del mismo. Los desarrollos de código abierto, por otro lado, se encuentran legalmente gobernados por licencias que dan ciertos derechos particulares en cuanto al uso y distribución del mismo.

En general, el software de código abierto debe mucho a tres organizaciones particulares: the Free Software Foundation (FSF), the Open Source Initiative (OSI), y the Creative Commons (CC). Cada una de ellas tiene una filosofía y un rol distinto dentro del mundo del código abierto. Existen, además, numerosas licencias open source específicas, de las cuales iremos hablando a lo largo del capítulo, y explicando en qué casos sería necesario utilizar una o la otra.

Explorando la protección de copyright y su relación con el software

Copyright es, como su indica su traducción del inglés, un reconocimiento legal de los derechos de copia que se aplican sobre cualquier cosa. En la mayoría de los países, si uno escribe un libro, saca una foto o escribe una aplicación, solo esa persona tiene los derechos para hacer copias del libro, la foto o la aplicación. De todas maneras, existe la posibilidad de dar derechos para hacer esas copias a terceros o incluso delegar completamente esos derechos a otra persona.

Las leyes de copyright, varían entre un país y otro, pero la mayoría firmó un acuerdo que obliga a los países a reconocer las leyes de copyright del resto. Eso quiere decir que, si una persona escribe un libro en Estados Unidos, esa persona tiene los derechos de copyright en Estados Unidos, pero también en Argentina, Islandia, o cualquier otro país que haya formado parte del acuerdo.

Dado que la mayoría de las leyes de copyright fueron escritas mucho antes de que existieran las primeras computadoras, generalmente no se adecúan a las necesidades de las mismas. Por ejemplo, las leyes de copyright prohíben originalmente la copia de algún desarrollo. En la mayoría de los casos relacionados al software, para poder hacer funcionar una aplicación, es necesario hacer una copia de la misma, por ejemplo:

- Es necesario hacer una copia del programa desde el medio de instalación al disco rígido para poder ejecutar una aplicación o al momento de la instalación.
- Una copia del programa, desde el disco rígido a la memoria principal se realiza cada vez que se ejecuta el mismo.

- Una copia del programa se realiza desde la memoria RAM al disco rígido, cada vez que el sistema operativo necesita ejecutar una operación de swap.
- Copias de un programa se realizan desde la memoria RAM hacia la memoria caché.
- Una copia de un programa puede hacerse cada vez que se realiza un backup.

En el pasado, estas copias estaban exentas de caer bajo la mirada de una ley de copyright, pero aun así en el sentido estricto, estas copias eran ilegales. Hoy en día, la mayoría de las leyes de copyright se modificaron para poder hacer “legales” estas copias que son necesarias para el funcionamiento de cualquier software.

Aunque el software esté sujeto a las leyes de copyright, la mayoría se distribuye junto con una licencia, que es el documento legal que modifica los derechos adquiridos por las leyes de copyright. En la mayoría de los casos uno no firma estas licencias, pero sí las acepta al momento de realizar la instalación de un determinado programa. Las licencias de software pueden modificar los derechos de las leyes de copyright en mayor o menor medida. Por ejemplo, la licencia GPL (General Public License), licencia utilizada por Linus Torvald en el desarrollo de Linux, modifica las leyes y da el derecho a redistribuir tanto los archivos binarios como el código fuente de cualquier aplicación.

Como regla general, las licencias de software propietario son aquellas que restringen aún más los derechos adquiridos por las leyes de copyright, mientras que las licencias de código abierto son aquellas que dan más libertades que las establecidas por las leyes de copyright.

The Free Software Foundation y los inicios del código abierto

En 1983, Richard Stallman inicia el proyecto GNU, con el fin de escribir un sistema operativo completo, basado en Unix, pero a diferencia del mismo no tenga restricciones en cuanto al uso del código fuente. Detrás del inicio de GNU y como contexto del mismo se inicial el grupo de Free Software Foundation, que establecerá una definición concreta de qué es el software libre y cuáles son las libertades del mismo.



Figura 1. Richard Stallman, fundador del proyecto GNU y el movimiento Free Software Foundation

¿Qué es el software libre?

El software libre es aquel que expone su código y no establece restricciones. Es importante destacar que el software libre no necesariamente es gratuito. Algunos ejemplos serían, por ejemplo, el sistema operativo Red Hat. El código del mismo es libre, por lo que cada persona puede compilarlo cuantas veces quiera. De todas maneras, el sistema operativo ya compilado, es comercializado por la empresa Red Hat Inc. Que además provee soporte para el mismo. La empresa asegura el funcionamiento del sistema operativo que ellos venden, si cualquier persona quiere compilar el suyo, es libre de hacerlo, pero no tendrá los beneficios que la empresa comercializa.

Las libertades del software

El grupo Free Software Foundation, establece cuatro libertades del software, estas son:

- Libertad de usar el software para cualquier propósito
- Libertad de examinar el código fuente y modificarlo para adecuarse a las necesidades
- Libertad de redistribuir el software
- Libertad de redistribuir el software modificado

En un mundo ideal, según Free Software Foundation, todo el software debe ser distribuido libremente con el código fuente y las cuatro libertades aseguradas. Si bien algunas distribuciones de GNU/Linux cumplen con estas ideas, existen algunas que incluyen software propietario. Tal es el caso de Ubuntu, que a pesar de ser un SO operativo basado en GNU/Linux y de código abierto, al momento de la instalación deja en manos del usuario la instalación de software propietario (generalmente códec específicos de audio y video, drivers para los dispositivos, etc.).

El punto fuerte del software de código abierto radica en que, por ejemplo, un usuario puede

encontrar una aplicación que se ajuste “casi” a sus necesidades. Siempre y cuando esa aplicación sea de código abierto, el usuario podrá modificarla para que se ajuste completamente a sus necesidades, y luego si lo desea, podrá redistribuir la aplicación modificada para ayudar a cualquiera que hubiera estado en la misma situación.

La filosofía que promueve la Free Software Foundation y las licencias inspiradas en ella, son comúnmente llamadas copyleft, ya que en vez de limitar la redistribución del contenido como definen las leyes de copyright, dan libertades de redistribuir y hacer prácticamente cualquier cosa con los contenidos.

The Free Software Foundation y la licencia GPL

Richard Stallman y la organización Free Software Foundation ya habían definido las libertades del software libre y qué es el software libre, pero no fue hasta que se creó la licencia GPL, que estas definiciones tuvieron un marco legal.

GPL (General Public License) es la expresión legal de los principios que establece la Free Software Foundation. Existen dos versiones principales de la licencia GPL: GPL Versión 2 y versión 3 (la versión 1 cayó prácticamente en desuso). Estas dos versiones de la licencia GPL, permiten aplicar las cuatro libertades del software libre a cualquier desarrollo que se realice. Además, establecen explícitamente que cualquier desarrollo licenciado bajo GPL, también deben ser licenciados bajo GPL. Esto quiere decir que cualquier aplicación que surja de la modificación de un software bajo licencia GPL, también debe poseer la licencia GPL, sin excepción.

Virtualización

***Arquitectura y Sistemas Operativos.
Tecnicatura Superior en Programación.
UTN-FRA***

Autores: Prof. Martín Isusi Seff

Revisores: Prof. Marcos Pablo Russo

Versión : 1



Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

¿Qué es la virtualización?

En informática, cuando se habla de virtualización, se hace referencia a la abstracción de algunos componentes físicos, en componentes lógicos. Si esta técnica puede aplicarse a distintos componentes (redes, distintos dispositivos electrónicos, etc.), en esta unidad nos enfocaremos a la virtualización completa de una computadora.

El primer antecedente de la virtualización de una computadora data de los años '60 en IBM. Luego, en el año 1974 surge el primer artículo que define formalmente los requerimientos necesarios para virtualizar una computadora moderna (el término "moderna" hace referencia a la tercera generación de computadoras). En esa definición, una máquina virtual puede virtualizar todos los recursos de hardware, incluyendo el procesador, la memoria, el almacenamiento y hasta la conexión de red. Para poder realizar esto, debe existir un monitor de máquina virtual, comúnmente llamado hypervisor, que será el encargado de proveer el entorno necesario para la operación de la máquina virtual.

De acuerdo a esta definición publicada en 1974, el hypervisor tiene que tener las siguientes características:

- **Fidelidad.** El entorno creado por el hypervisor debe ser idéntico al de una máquina física.
- **Aislamiento.** El hypervisor debe tener control completo de la máquina que está virtualizando, y debe aislar la misma del sistema que la está ejecutando.
- **Rendimiento.** No debe haber, o en su defecto, debe haber poca diferencia de rendimiento entre una máquina virtual y su equivalente físico.

A pesar de que una máquina virtual debe cumplir con las tres características, es común que sólo cumplan con las primeras dos y no con la característica de rendimiento.

Entendiendo la importancia de la virtualización

En los últimos tiempos hubo un gran avance en lo que respecta al desarrollo de nuevas tecnologías que traen aparejadas mejoras de rendimiento y la posibilidad de realizar tareas de forma paralela, sin la necesidad de simular ese paralelismo aplicando pequeños instantes de tiempo a cada operación. Por otro lado, además del avance de estas tecnologías, también hubo un gran avance dentro de las tecnologías de información asociadas al software y a las redes. Internet creció exponencialmente. Esto trajo aparejado un volumen de datos inmenso y fue necesaria una enorme infraestructura de hardware para poder manejar este volumen de datos. Esto se traduce, básicamente, en que se necesitaban muchas computadoras. Desafortunadamente, este hardware no es gratuito y de hecho muchas veces es costoso, además de que ocupa mucho espacio y una infraestructura edilicia propicia, lo cual hace que los costos se eleven aún más.

Si resumimos esta situación, podemos decir que necesitamos muchas computadoras para que

los sistemas como hoy los conocemos funcionen. Para eso necesitamos espacio, y la capacidad de afrontarlos económicamente. Pero es por todo esto que surge la alternativa de la virtualización. Sabiendo que la virtualización puede simular una computadora lógica (que hace lo mismo que una computadora física), dentro de una máquina física, podríamos tener una sola computadora física, que virtualice más de una computadora lógica. Allí radica la principal ventaja de la virtualización.

En un principio, la virtualización se limitaba a servidores. Hoy en día con distintos softwares (algunos de ellos libres), podemos virtualizar utilizando una computadora de escritorio.

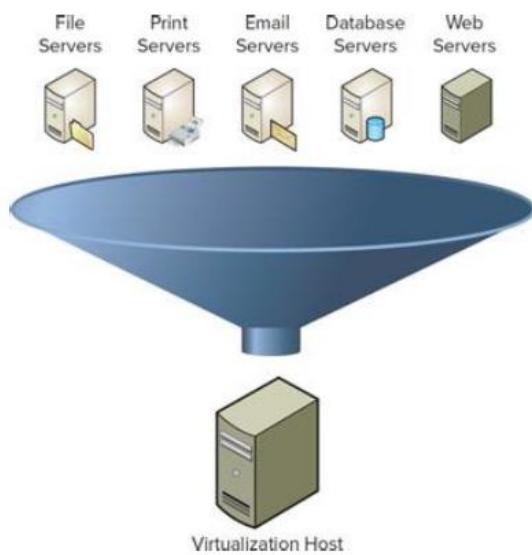


Figura 1. Esquema representativo de un host (máquina física), que virtualiza distintos tipos de servidores.

Definición y funciones del Hypervisor

El Hypervisor, anteriormente llamado monitor de máquina virtual, es una capa de software que reside entre la máquina virtual y el hardware del host¹.

Sin el Hypervisor, las máquinas virtuales (para ser más exactos, el sistema operativo de cada máquina virtual) tratarían de comunicarse directamente con el hardware del host, haciendo que se produzca un caos entre las mismas y el sistema operativo del host. El Hypervisor se encargará de manejar las interacciones entre las máquinas virtuales y el hardware del host.

¹ De ahora en más, cuando se nombra al *host*, se estará haciendo referencia al equipo físico que contiene una o más máquinas virtuales.

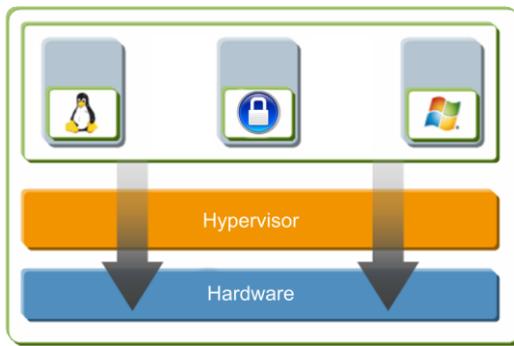


Figura 2. Esquema del sistema con el Hypervisor y distintas máquinas virtuales con diferentes sistemas operativos.

Hypervisor tipo 1

El Hypervisor tipo 1 es aquel que se ejecuta directamente sobre el hardware del host, sin un sistema operativo en el medio. Al no tener un sistema operativo como intermediario, el Hypervisor tipo 1 puede tener una interacción directa con el hardware, haciendo que tenga un rendimiento mejor que los Hypervisores tipo 2.

Además de la ventaja del rendimiento, se considera que este tipo de Hypervisor es más seguro que los de tipo 2. Al no existir un sistema operativo intermediario, no existirá la posibilidad de que este afecte de algún modo la ejecución del sistema virtualizado.

Por último, este tipo de Hypervisor requiere de menos procesamiento (porque no existe un sistema operativo intermedio que procese las interacciones). Esto permite que, utilizando el Hypervisor tipo 1, podamos ejecutar más máquinas virtuales al mismo tiempo.

Hypervisor tipo 2

El Hypervisor tipo dos es, básicamente, una aplicación que se ejecuta sobre el sistema operativo. Por lo tanto, con este tipo de Hypervisor, el sistema operativo del host hará de intermediario, manejando las interacciones entre la máquina virtual y el hardware.

Este tipo de Hypervisor tiene la ventaja de soportar un rango de hardware más amplio que el tipo 1, ya que hereda la compatibilidad del sistema operativo host. Además, gracias a esta "herencia" de compatibilidad, es más fácil la instalación y configuración de este tipo de máquinas virtuales.

El Hypervisor tipo 2 no es tan eficiente como el tipo 1 debido a la capa intermedia que representa el sistema operativo del host. Por ejemplo, si el sistema operativo dentro de la máquina virtual necesita escribir información en el disco, no podrá acceder a él directamente. Deberá comunicarse con el sistema operativo del host y este será el encargado de comunicarse con el disco. Lo mismo sucede con todos los recursos como la placa de red, de sonido, etc.

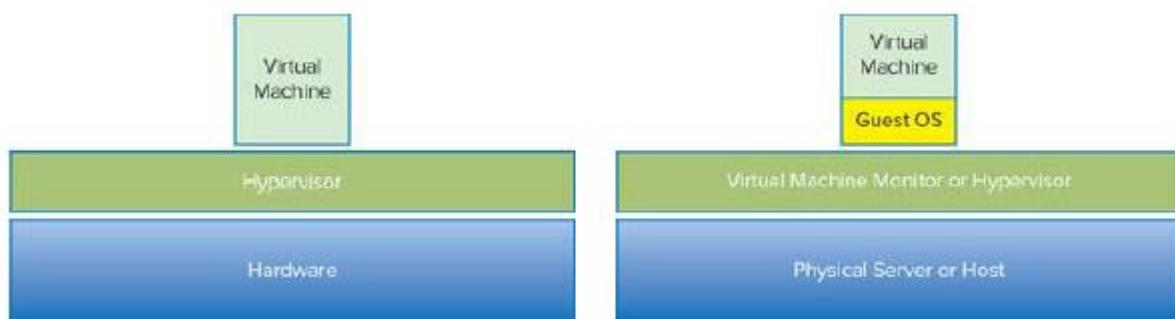


Figura 3. El esquema a la izquierda representa el Hypervisor tipo 1, mientras que el diagrama de la derecha el tipo 2. En amarillo, 'Guest OS' hace referencia al sistema operativo del host.

Línea de comandos y comandos principales

***Arquitectura y Sistemas Operativos.
Tecnicatura Superior en Programación.
UTN-FRA***

Autores: Prof. Martín Isusi Seff

Revisores: Prof. Marcos Pablo Russo

Versión: 1



Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

Entendiendo la línea de comandos

La línea de comandos, en GNU/Linux, Unix, OSX o Windows, es una terminal que permite al usuario ejecutar instrucciones en la computadora. La línea de comandos puede parecer complicada al principio, pero es una de las herramientas más potentes que tiene a mano cualquier desarrollador o administrador de sistemas.

Si bien la mayoría de los usuarios acostumbra a usar una interfaz gráfica para realizar cualquier acción, mediante la línea de comandos, en gran medida, se pueden realizar las mismas acciones.

Dependiendo del sistema operativo, la línea de comandos tendrá más o menos importancia. En Microsoft Windows, la línea de comandos prácticamente no es utilizada ya que el propio sistema operativo se construye bajo una arquitectura basada en "ventanas". Su base es una interfaz gráfica. Por otro lado, en GNU/Linux (cualquiera sea la distribución), la línea de comandos es quizás la herramienta más importante. Si bien hoy en día cualquier distribución orientada al usuario se instalará con una interfaz gráfica, la misma no hace más que ejecutar los comandos que se el usuario ejecutaría en la terminal de línea de comandos.

La principal ventaja de la línea de comandos radica en la facilidad que nos da a la hora de automatizar procesos extensos o repetitivos. En estos casos se podría, por ejemplo, crear un *script*¹, que ejecute paso a paso los comandos que de otra manera se ejecutarían a mano.

Existen distintos intérpretes de línea de comandos, bash, csh, cmd, etc.

Cada comando que ejecutamos en la terminal de línea de comandos no forma parte del intérprete en sí, sino que cada comando no es más que una aplicación almacenada en un directorio específico.

Tal como cualquier aplicación, los sistemas operativos tienen variables. Las variables, llamadas "variables de entorno", contienen información necesaria durante el funcionamiento del sistema. Desde la terminal de línea de comandos podemos ver el valor de las variables, así como también modificar el mismo. Cuando hablamos de terminal de línea de comandos, es importante mencionar particularmente una de las tantas variables de entorno, la variable PATH. Esta variable contiene rutas a directorios o enlaces. Estos directorios contienen los ejecutables de los comandos que ejecutamos desde la terminal línea de comandos. Si desde la terminal ejecuto el comando *ls*, el archivo ejecutable del mismo deberá estar dentro de alguno de los directorios especificados en la variable de entorno PATH.

¿Cómo se ve la terminal línea de comandos?

La terminal de línea de comandos, en cualquier sistema operativo, no es más que una pantalla (o una ventana si se abre desde una interfaz gráfica) que imprime texto. La terminal de línea

¹ Llamamos script al código fuente de una aplicación, sea el lenguaje que sea. En el contexto de un sistema operativo nos referimos a *script* al código que contiene comandos a ser ejecutados por el sistema operativo.

de comandos no muestra gráficos.

Cuando se inicia la terminal de línea de comandos, la misma se “ubicará” dentro de un directorio determinado (algo similar a lo que sucede cuando se abre un explorador de archivos gráfico) y quedará “esperando” que el usuario escriba y ejecute un comando.

El comando [quizás] más importante, es el comando **cd**. Este comando es el utilizado para cambiar de ubicación la línea de comandos. El comando *cd*, se utiliza escribiendo *cd ruta*, donde *ruta* es la *ruta relativa* o *ruta absoluta* a donde nos queremos ubicar.

```
alumno@alumno-VirtualBox:~$ cd Desktop/  
alumno@alumno-VirtualBox:~/Desktop$ █
```

Figura 1.

En la Figura 1 se ve una terminal de línea de comandos. En la primera línea de la misma vemos la palabra 'alumno'. La misma corresponde al nombre del usuario con el que se inició la sesión. Luego de '@' se encuentra el nombre del equipo. Seguido de ':' se encuentra la ruta en la que está ubicada la terminal. Aquí vemos que está ubicada en '~'. Esto quiere decir que la terminal está ubicada en la carpeta personal del usuario 'alumno'. El símbolo '\$' indica que se trata de un usuario no administrador. Y, por último, tenemos 'cd Desktop', que es el comando que se ejecutó. Con ese comando, la terminal se ubicará en el directorio 'Desktop' que está dentro de '~'. Nótese cómo en la segunda línea, luego de ':' ahora se muestra '~/Desktop'.

Rutas o paths

Antes de continuar con los comandos principales, es necesario mencionar el concepto de *ruta* o *path*. Una *ruta*, es la dirección de un archivo (o directorio) dentro del sistema de archivos. Cuando trabajamos en la terminal de línea de comandos, las *rutas* pueden representarse de dos maneras: *rutas absolutas* o *rutas relativas*.

Rutas absolutas

Las rutas absolutas son rutas que se escriben desde la raíz del sistema de archivos. Tomemos como ejemplo la siguiente estructura de directorios:

```
/  
|  
|__ Directorio 1  
|   |  
|   |__ Directorio 2  
|   |  
|   |__ Directorio 3
```

En la estructura tenemos un directorio raíz llamado **/** (en GNU/Linux, siempre el directorio raíz

es /. En Windows, por ejemplo, el directorio raíz es aquel conocido como C:/). Dentro del directorio raíz se encuentra **Directorio 1**, y dentro del mismo se encuentran **Directorio 2** y **Directorio 3**.

La ruta absoluta de **Directorio 3** sería: **/Directorio 1/Directorio 3**. Allí vemos que está la raíz (/), el Directorio 1 y luego, seguido por / (en este caso no representa la raíz), se encuentra el Directorio 3.

Rutas relativas

Las rutas relativas son aquellas que no se indican desde el directorio raíz, sino que se escriben desde la “posición actual”. Supongamos que abrimos una terminal de línea de comandos que se encuentra ubicada en **Directorio 1**. La ruta absoluta de **Directorio 3** sería la mencionada previamente, pero la ruta absoluta sería simplemente escribir **Directorio 3**. ¿Por qué? Porque **Directorio 3** se encuentra dentro de la posición donde estamos ubicados. Analicemos el siguiente caso:

```

/
|
|__ Directorio 1
    |
    |__ Directorio 2
    |
    |__ Directorio 3
        |
        |__ Directorio 4

```

Ahora, dentro de **Directorio 3**, se encuentra **Directorio 4**. Si nos encontramos con la terminal dentro de **Directorio 1**, ¿Cuál sería la ruta relativa hacia **Directorio 4**? Primero tenemos que ver que **Directorio 4**, está dentro de **Directorio 3**. Para ir desde nuestra supuesta ubicación hacia **Directorio 3**, simplemente tenemos que escribir el nombre del mismo. Por último, como **Directorio 4** está dentro de **Directorio 3** escribiríamos lo siguiente: **Directorio 3/Directorio 4**.

Comando ls

El comando **ls** nos permite poder ver los contenidos de los directorios. Al este comando pueden aplicarse las siguientes opciones:

-l	Muestra los archivos en forma de columna.
-a	Muestra todos los archivos (incluyendo los ocultos)
-R	Listado recursivo de todos los archivos y subdirectorios.
-i	Muestra el número de inodo.
-h	Muestra el espacio ocupado del archivo en MB, Kbyte, etc.
-t	Ordena por la fecha y hora de modificación.

Ejemplos:

```
alumno@alumno-VirtualBox:~$ ls -l
total 44
drwxrwxr-x 4 alumno alumno 4096 ago 11 13:53 Datos
drwxrwxr-x 6 alumno alumno 4096 ago 11 13:52 Datos_Generales
drwxr-xr-x 2 alumno alumno 4096 ago  9 14:18 Desktop
drwxr-xr-x 2 alumno alumno 4096 ago 11 13:25 Documents
drwxr-xr-x 2 alumno alumno 4096 ago  9 14:18 Downloads
drwxr-xr-x 2 alumno alumno 4096 ago  9 14:18 Music
drwxrwxr-x 3 alumno alumno 4096 ago 11 13:50 NuevoDirectorio
drwxr-xr-x 2 alumno alumno 4096 ago  9 14:18 Pictures
drwxr-xr-x 2 alumno alumno 4096 ago  9 14:18 Public
drwxr-xr-x 2 alumno alumno 4096 ago  9 14:18 Templates
drwxr-xr-x 2 alumno alumno 4096 ago  9 14:18 Videos
alumno@alumno-VirtualBox:~$ █
```

Figura 2

Obtener información de los comandos

Gracias a una serie de comandos podemos obtener información de los archivos de configuración, y de los comandos que ejecutamos, también tendremos otros comandos que nos permite realizar búsquedas de manuales. El comando que utilizaremos es apropos. Cada manual contiene una pequeña descripción, este comando nos permite buscar dentro de esas descripciones. Las opciones que podemos usar con este comando son las siguientes:

-e, --extract	Cada palabra clave se comparará de forma exacta con los nombres de páginas y las descripciones.
-d, --debug	Imprime información de depuración.
-r, --regex	Interpreta cada palabra clave como una expresión regular. Este comportamiento es el predeterminado. Cada palabra clave se comparará con los nombres de las páginas y las descripciones.
-w, --wildcard	Idem a la opción -r, --regex , aunque además permite la utilización de comodines.
-M ruta, --manpath=ruta	Especifica rutas separadas por "dos puntos" de la ubicación de los manuales. Por defecto apropos utiliza la variable de entorno \$MANPATH en todo caso que esté vacía la variable de entorno, utiliza como ruta para los manuales la variable \$PATH
-h, --help	Ayuda

Ejemplos:

```
alumno@alumno-VirtualBox:~$ apropos cd
apt-cdrom (8)           - APT CD-ROM management utility
cd-create-profile (1)    - Color Manager Profile Creation Tool
cd-fix-profile (1)       - Color Manager Testing Tool
cd-it8 (1)               - Color Manager Testing Tool
hex2hcd (1)              - firmware converter
hipercdecode (1)         - Decode a HIPERC stream into human readable form.
rsyncd.conf (5)          - configuration file for rsync in daemon mode
sbigtogm (1)              - convert an SBIG CCDOPS file into a portable graymap
systemd-timesyncd (8)     - Network Time Synchronization
systemd-timesyncd.service (8) - Network Time Synchronization
tcdrain (3)              - get and set terminal attributes, line control, get and set baud rate
timesyncd.conf (5)        - Network Time Synchronization configuration files
timesyncd.conf.d (5)       - Network Time Synchronization configuration files
xburn (1)                 - Simple CD/DVD burning tool
alumno@alumno-VirtualBox:~$
```

Figura 3

Comando *man*

Con el comando *man* podemos ver los manuales de los programas y los archivos de configuración. La siguiente tabla muestra los números de sección del manual y los tipos de páginas que contienen.

1	Programas ejecutables y guiones del intérprete de comandos.
2	Llamadas del sistema (funciones servidas por el núcleo).
3	Llamadas de la biblioteca (funciones contenidas en las bibliotecas del sistema).
4	Ficheros especiales (generalmente en /dev).
5	Formato de ficheros y convenios (por ejemplos, /etc/passwd).
6	Juegos.
7	Paquetes de macros y convenios.
8	Órdenes de administración del sistema (generalmente solo son para root)
9	Rutinas del núcleo.

Con el comando *man*, podemos usar las siguientes opciones:

-d, --debug	Imprime información de depuración.
-t comando	Formatea la página del manual del comando en postscript.
-k comando	Ídem a apropos .
-f comando	Busca las páginas del manual referenciadas por <i>comando</i> , e imprime la descripción de las que encuentra. Equivalente al comando whatis .
-u, --update	Los cachés de los índices de las bases de datos son actualizados sobre

	la marcha, es decir, no es necesario que mandb se ejecute periódicamente para mantener la consistencia.
-M ruta --manpath=ruta	Especifica rutas separadas por "dos puntos" de la ubicación de los manuales. Por defecto man utiliza la variable de entorno \$MANPATH en todo caso que esté vacía la variable de entorno, utiliza como ruta para los manuales la variable \$PATH .
-P página --pager=página	La <i>página</i> es el tamaño, esa sentencia sobreescribe la variable de entorno \$PAGER .
-d, --debug	Imprime información de depuración.

Comando *info*

Este comando supera la información del comando *man*, y nos permite navegar por medio de los enlaces como si fuera una página web. Las opciones que podemos usar con el comando *info* son las siguientes:

-k, --aprops	Buscar cadena en todos los índices de todos los manuales.
-d --directory=DIR	Agrega un directorio <i>DIR</i>
-f --file=ARCHIVO	Se especifica un archivo determinado <i>ARCHIVO</i>
-M ruta --manpath=ruta	Especifica rutas separadas por "dos puntos" de la ubicación de los manuales. Por defecto man utiliza la variable de entorno \$MANPATH en todo caso que esté vacía la variable de entorno, utiliza como ruta para los manuales la variable \$PATH .
-h, --help	Muestra ayuda del comando.

Comando *whatis*

El comando *whatis* nos da una pequeña descripción de un comando y las secciones de *man* que podemos consultar. Las opciones que podemos usar con el comando *whatis* son las siguientes:

-d, --debug	Imprime información de depuración.
-r, --regex	Interpreta cada palabra clave como una expresión regular. Este comportamiento es el predeterminado. Cada palabra clave se comparará con los nombres de las páginas y las descripciones.
-w, --wildcard	Idem a la opción -r, --regex , aunque además permite la utilización de

	comodines.
-M ruta, --manpath=ruta	Especifica rutas separadas por "dos puntos" de la ubicación de los manuales. Por defecto apropos utiliza la variable de entorno \$MANPATH en todo caso que esté vacía la variable de entorno, utiliza como ruta para los manuales la variable \$PATH
-h, --help	Ayuda

Creación de un archivo vacío o cambio de fecha y hora: **touch**

Este comando nos sirve para dos motivos: Si un archivo no existe este comando lo creará con el nombre especificado y con la fecha del momento. Por otro lado, si el archivo existe, le cambiará la fecha y hora. Las opciones que podemos utilizar con el comando *touch* son:

-a	Cambia solamente el tiempo de acceso.
-d string --date=string	Se cambia a la fecha que se indica en <i>string</i> .
-m	Cambia solamente la hora.

Ejemplo:

```
alumno@alumno-VirtualBox:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
alumno@alumno-VirtualBox:~$ touch nuevo_archivo
alumno@alumno-VirtualBox:~$ ls
Desktop Documents Downloads Music nuevo_archivo Pictures Public Templates Videos
alumno@alumno-VirtualBox:~$ █
```

Figura 4

Copia de archivos: **cp**

El comando *cp* nos permite copiar un archivo o directorio completo. Para usarlo escribimos:

cp [opciones] origen destino

Las opciones que podemos usar con este comando son:

-a	Es la combinación de -d (mantiene los enlaces) y -R (recursivo).
-i	Interactivo. Preguntará por cada archivo que se está copiando.
-p	Mantiene los permisos, dueño y grupo de los archivos/directorios.
-v	Mostrará información de los archivos/directorios que se están copiando (verbose).

Ejemplos:

```

alumno@alumno-VirtualBox:~$ ls
Desktop Documents Downloads Music nuevo_archivo Pictures Public Templates Videos
alumno@alumno-VirtualBox:~$ cp nuevo_archivo nuevo_archivo_2
alumno@alumno-VirtualBox:~$ ls
Desktop Downloads nuevo_archivo Pictures Templates
Documents Music nuevo_archivo_2 Public Videos
alumno@alumno-VirtualBox:~$ █
    
```

Figura 5

Para especificar el origen y destino de la copia, es posible utilizar comodines como * o ?.

```

alumno@alumno-VirtualBox:~$ ls
Desktop Downloads nuevo_archivo Pictures Templates
Documents Music nuevo_archivo_2 Public Videos
alumno@alumno-VirtualBox:~$ cp nuevo_* Documents/
alumno@alumno-VirtualBox:~$ ls Documents/
nuevo_archivo nuevo_archivo_2
alumno@alumno-VirtualBox:~$ █
    
```

Figura 6

Mostrar contenido de archivos: **more**, **less** y **cat**

Tanto el comando **more** y **less** nos muestra el contenido del archivo y solo podemos bajar de a una línea por vez (con la tecla enter) o avanzar por página (con la tecla space). El comando **less** debe ser instalado como paquete apt, usando el comando **apt-get install less**. Además, el comando **less** nos permite realizar búsquedas dentro del archivo. También con las teclas (pgdown) podemos bajar a la página siguiente y con la tecla (pgup) podemos volver a la página anterior.

El comando **cat**, cumple la misma función que los comandos **more** o **less**, con la diferencia que el contenido completo se muestra en la pantalla.

Ejemplo:

```

alumno@alumno-VirtualBox:~$ ls
Desktop Documents Downloads Music Pictures Public saludo.txt Templates Videos
alumno@alumno-VirtualBox:~$ cat saludo.txt
Hola mundo!
alumno@alumno-VirtualBox:~$ █
    
```

Figura 7

Redirección: >, >>, <

A través de la redirección, podemos tomar la salida de un programa y enviarla automáticamente a un archivo. Este proceso lo maneja la propia shell, en un lugar del

programa. La redirección se divide en tres clases: salida a un archivo, añadir al final de un archivo, o envió de un archivo como entrada. Para recoger la salida de un programa en un archivo, finalice la línea del comando con el símbolo mayor que (>) y el nombre del archivo en el cual quiere guardar la salida redirigida.

```
alumno@alumno-VirtualBox:~$ ls
Desktop Documents Downloads Music Pictures Public saludo.txt Templates Videos
alumno@alumno-VirtualBox:~$ ls >> listaDeArchivo.txt
alumno@alumno-VirtualBox:~$ ls
Desktop Downloads Music Public Templates
Documents listaDeArchivo.txt Pictures saludo.txt Videos
alumno@alumno-VirtualBox:~$ cat listaDeArchivo.txt
Desktop
Documents
Downloads
listaDeArchivo.txt
Music
Pictures
Public
saludo.txt
Templates
Videos
alumno@alumno-VirtualBox:~$
```

Figura 8

Utilización de pipes (|)

Los pipes son un mecanismo por el cual la salida de un programa se puede enviar como entrada de otros programas. Los programas individuales se pueden encadenar juntos para convertirse en unas herramientas extremadamente potentes.

Ejemplo:

```
alumno@alumno-VirtualBox:~$ ls -la /etc/ | more
total 1120
drwxr-xr-x 128 root root 12288 ago 11 12:51 .
drwxr-xr-x 24 root root 4096 ago 11 12:51 ..
drwxr-xr-x 3 root root 4096 abr 20 2016 acpi
-rw-r--r-- 1 root root 3028 abr 20 2016 adduser.conf
drwxr-xr-x 2 root root 4096 ago 11 12:48 alternatives
-rw-r--r-- 1 root root 401 dic 28 2014 anacrontab
drwxr-xr-x 6 root root 4096 abr 20 2016 aptm
drwxr-xr-x 3 root root 4096 ago 11 12:50 apparmor
drwxr-xr-x 8 root root 4096 ago 11 12:51 apparmor.d
drwxr-xr-x 5 root root 4096 ago 11 12:50 apport
-rw-r--r-- 1 root root 389 abr 18 2016 appstream.conf
drwxr-xr-x 6 root root 4096 ago 9 14:10 apt
drwxr-xr-x 2 root root 4096 abr 20 2016 at-spi2
drwxr-xr-x 3 root root 4096 abr 20 2016 avahi
-rw-r--r-- 1 root root 2188 ago 31 2015 bash.bashrc
-rw-r--r-- 1 root root 45 ago 12 2015 bash_completion
drwxr-xr-x 2 root root 4096 ago 11 12:50 bash_completion.d
-rw-r--r-- 1 root root 367 ene 27 2016 bindresvport.blacklist
drwxr-xr-x 2 root root 4096 abr 12 2016 binfmt.d
drwxr-xr-x 2 root root 4096 abr 20 2016 bluetooth
-rw-r--r-- 1 root root 33 abr 20 2016 brlapi.key
drwxr-xr-x 7 root root 4096 abr 20 2016 brltty
-rw-r--r-- 1 root root 23444 abr 11 2016 brltty.conf
drwxr-xr-x 3 root root 4096 abr 20 2016 ca-certificates
-rw-r--r-- 1 root root 7788 abr 20 2016 ca-certificates.conf
drwxr-xr-x 2 root root 4096 abr 20 2016 calendar
drwxr-s--- 2 root dip 4096 abr 20 2016 chatscripts
drwxr-xr-x 2 root root 4096 abr 20 2016 console-setup
drwxr-xr-x 2 root root 4096 abr 20 2016 cron.d
```

Figura 9

En el caso que se quiera concatenar comandos, se pueden aplicar dos opciones: Separar los comandos con el carácter ; o con &&. Si se utiliza ;, no importa si el primer comando resulta en un error, el segundo de cualquier manera se ejecutará. En caso de que se separen los comandos con &&, si el primer comando termina en error, el segundo no se ejecutará.

Ejemplo:

```
alumno@alumno-VirtualBox:~$ ls /noExiste && ls
ls: cannot access '/noExiste': No such file or directory
alumno@alumno-VirtualBox:~$ ls /noExiste ; ls
ls: cannot access '/noExiste': No such file or directory
Desktop   Downloads      Music   Public   Templates
Documents listaDeArchivo.txt Pictures  saludo.txt Videos
alumno@alumno-VirtualBox:~$
```

Figura 10

Eliminar archivos o directorios: rm

El comando **rm** nos permite borrar tanto archivos como directorios completos. Las opciones que se pueden utilizar son:

-i	Preguntará si se desea borrar el archivo o no.
-f	En forma forzada borra sin importar si contiene o no archivos.
-r i -R	Borra de forma recursiva tanto archivos como directorios.

-v	Muestra qué se está borrando (verbose).
-----------	---

Si como root (es decir super usuario) ejecutamos el comando **rm - rf** / nos borrara casi todo el sistema operativo, haciendo que el sistema deje de funcionar.

Mover archivos/directorios o renombrarlos: **mv**

El comando **mv** nos permite tanto mover como renombrar archivos o directorios. Las opciones que podemos utilizar son:

-i	Preguntará si se desea borrar el archivo o no.
-f	En forma forzada borra sin importar si contiene o no archivos.
--backup	Creará un backup del archivo antes de ser movido.
-v	Muestra qué se está borrando (verbose).

Creación de un directorio: **mkdir**

Utilizamos el comando **mkdir** cuando necesitamos crear un nuevo directorio. Las opciones que podemos utilizar son:

-p	Creará tanto el directorio padre como los subdirectorios.
-m --mode=MODE	Permite crear un directorio/subdirectorio con permisos específicos.
-v	Muestra qué es lo que se está creando (verbose).

Ejemplo²:

```
alumno@alumno-VirtualBox:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
alumno@alumno-VirtualBox:~$ mkdir -p NuevoDirectorio/OtroDirectorio
alumno@alumno-VirtualBox:~$ ls
Desktop Documents Downloads Music NuevoDirectorio Pictures Public Templates Videos
alumno@alumno-VirtualBox:~$ tree NuevoDirectorio/
NuevoDirectorio/
└── OtroDirectorio

1 directory, 0 files
alumno@alumno-VirtualBox:~$
```

Figura 11

En el siguiente ejemplo, se crean los directorios “Archivos”, “Documentos”, “Fotos” e “Imagenes” dentro del directorio “Datos_Generales”:

² En el ejemplo se utilizará también el comando **tree**. El mismo no forma parte de los comandos básicos de GNU/Linux y debe ser instalado aparte. El mismo muestra la estructura jerárquica del comando que indiquemos.

```
alumno@alumno-VirtualBox:~$ mkdir -p Datos_Generales/{Archivos,Documentos,Fotos,Imagenes}
alumno@alumno-VirtualBox:~$ tree Datos_Generales/
Datos_Generales/
└── Archivos
    └── Documentos
        └── Fotos
            └── Imagenes

4 directories, 0 files
alumno@alumno-VirtualBox:~$
```

Figura 12

En el siguiente ejemplo se crean los directorios “Datos1” y “Datos2” dentro del directorio “Datos”. Además, dentro de los directorios “Datos1” y “Datos2”, se crea un directorio “Datos_Generales”:

```
alumno@alumno-VirtualBox:~$ mkdir -p Datos/{Datos1,Datos2}/Datos_Generales
alumno@alumno-VirtualBox:~$ tree Datos
Datos
└── Datos1
    └── Datos_Generales
└── Datos2
    └── Datos_Generales

4 directories, 0 files
alumno@alumno-VirtualBox:~$
```

Figura 13

Remover un directorio: *rmdir*

Utilizamos ***rmdir*** para eliminar directorios que estén vacíos. La única opción que podemos utilizar con ***rmdir*** es **-p**, que eliminará también la carpeta superior (parent).

Borrar la pantalla

Para borrar la pantalla, utilizaremos el comando ***clear***. En algunos casos es posible utilizar la combinación de teclas ***ctrl+I***.

Expresiones regulares

Las expresiones regulares son generalmente cadenas de caracteres, que se utilizan para representar patrones de texto. Cada carácter en una expresión regular puede representar tanto un carácter específico como un conjunto de caracteres.

Estas expresiones son muy utilizadas, ya que permiten, por ejemplo, representar patrones para realizar búsquedas, validar datos que ingrese el usuario, etc. En este curso, utilizaremos las expresiones regulares para realizar búsquedas en archivos, eliminar archivos que comiencen con un determinado patrón, y algunos usos más.

A continuación, se muestra un cuadro con las expresiones regulares, y qué representan las mismas:

?	El elemento precedente es opcional y debe coincidir al menos una vez.
*	El elemento precedente debe coincidir cero o más veces.
{n}	El elemento precedente debe coincidir exactamente n veces.
+	El elemento precedente debe coincidir una o más veces.
{,m}	El elemento precedente es opcional y debe coincidir al menos m veces.
{n,m}	El elemento precedente debe coincidir al menos n veces, pero no más de m veces.
Pablo	Busca la cadena Pablo dentro del texto.
^	La expresión posterior debe ser el inicio de una línea.
\$	La expresión precedente debe ser el final de una línea.
[mn]	El elemento debe ser m o n
[^mn]	El elemento no debe ser ni m ni n
.	El elemento puede ser cualquier carácter.
\	Carácter de escape. Por ejemplo, el símbolo . indica cualquier carácter, si aplicamos \. Hacemos referencia al carácter .
[a-z]	Representa cualquier carácter de la a a la z minúscula.
\t	Representa el carácter TAB .
\r	Representa el carácter "retorno de carro".
\n	Representa el carácter "nueva línea".
\a	Representa una "campana" o "beep". Carácter que produce un sonido cuando se imprime.
\e	Representa la tecla escape .
\f	Representa un salto de página.
\v	Representa un tabulador vertical.
\x	Se utiliza para representar caracteres ASCII o ANSI si se conoce su código. De esta forma, si se busca el símbolo de derechos de autos y la fuente en la que se busca utiliza el conjunto de caracteres Latin-1, es posible encontrarlo utilizando \xA9 .
\u	Se utiliza para representar caracteres Unicode si se conoce su código. \u00A2 representa el símbolo de centavos. No todos los motores de expresiones regulares soportan Unicode.
\d	Representa cualquier dígito del 0 al 9.
\w	Representa cualquier carácter alfanumérico.
\s	Representa un espacio en blanco.
\D	Representa cualquier carácter que no sea un dígito del 0 al 9.
\W	Representa cualquier carácter no alfanumérico.

\S	Representa cualquier carácter que no sea un espacio en blanco.
\A	Representa el inicio de la cadena. No un carácter, sino una posición.
\Z	Representa el final de la cadena. No un carácter, sino la posición.
\b	Marca el inicio y el final de una palabra.
\B	Marca la posición entre dos caracteres alfanuméricos o dos no-alfanuméricos.

Búsqueda en archivos utilizando expresiones regulares

El comando **grep** toma una expresión regular de la línea de comandos, lee la entrada estándar o una lista de archivos, e imprime las líneas que contengan coincidencias para la expresión regular.

Las opciones disponibles del comando **grep** son:

-c	Modificar la salida normal del programa. En lugar de imprimir por salida estándar las líneas coincidentes, imprime la cantidad de líneas que coincidieron en cada archivo.
-e PATRÓN	Usar PATRÓN como patrón de búsqueda.
-f ARCHIVO	Obtener los patrones del ARCHIVO .
-H	Imprime el nombre del archivo con cada coincidencia.
-r	Busca recursivamente dentro de todos los subdirectorios del directorio actual.
-i	Busca mayúsculas o minúsculas.
-w	Buscar la palabra completa.
-v	Buscar lo contrario.
-l	Muestra solo los archivos que contengan la palabra buscada.
-c	Cuenta la cantidad de ocurrencias.
-E	Utiliza expresiones regulares extendidas. En vez de usar grep , en este caso se podría usar egrep .
-o	Muestra solamente o que queremos buscar en la línea.

Modificar archivos utilizando el comando sed

El comando **sed** permite, de una manera simple, modificar un archivo, sustituyendo líneas y reemplazándolas por otras. Para poder indicar qué líneas sustituir, se utilizan, también, expresiones regulares.

Las opciones que se pueden utilizar con el comando **sed** son:

-c	Modificar la salida normal del programa. En lugar de imprimir por salida estándar las líneas coincidentes, imprime la cantidad de líneas que coincidieron en cada archivo.
-----------	--

-n	Suprime la muestra automática del espacio de patrones.
--quiet	
--silence	
-e order	Agrega <i>order</i> a la lista de órdenes a ejecutar.
--expresion=order	
-f archivo	Agrega el contenido del fichero <i>archivo</i> a la lista de órdenes a ejecutar.
--file=archivo	
--posix	Desactiva todas las extensiones de GNU
-r	Utiliza expresiones regulares extendidas en la expresión.
--regexp-extended	
-s	Considera los archivos como separados en lugar de un solo flujo continuo.
--separate	
-u	Carga cantidades mínimas de datos de los archivos de entrada y vacía los almacenamientos temporales de salida con más frecuencia.
--unbuffered	
--help	
--version	

Ejemplo. Comando grep para obtener mails de un archivo.

1. Buscamos un comienzo de palabra **\b** que contenga uno o más caracteres **+** dentro del conjunto **A-Z, 0-9, ., _, % y -**, que son los caracteres admitidos para un nombre de usuario. Si unimos todas estas condiciones, la expresión regular que obtenemos es: **\b[A-Z0-9._%-]+**
2. En una dirección de email, luego del nombre de usuario vendrá el carácter **@**, por lo que nuestra expresión regular tomará la siguiente forma: **\b[A-Z0-9._%-]+@[A-Z0-9._%-]+@**
3. Luego del carácter **@**, tendremos el dominio. El dominio se forma por uno o más caracteres de la **A-Z, 0-9, . y -**. Por lo tanto nuestra expresión ahora será **\b[A-Z0-9._%-]+@[A-Z0-9._%-]+@[A-Z0-9.-]+**
4. Por último, luego del dominio tendremos un **punto (.)**, seguido por **2, 3 o 4 caracteres de la A a la Z**. Esto último, además, debe ser el **final de una palabra**. La expresión final será: **\b[A-Z0-9._%-]+@[A-Z0-9.-]+@[A-Z]{2,4}**

Para el ejemplo, crearemos un archivo llamado mail.txt y le agregaremos dos líneas. En cada una de ellas habrá una dirección de mail.

```
alumno@alumno-VirtualBox:~$ echo "El mail de Paul es paul@hotmail.com" > mail.txt
alumno@alumno-VirtualBox:~$ echo "El mail de Ringo es ringo@hotmail.com" >> mail.txt
alumno@alumno-VirtualBox:~$ egrep -oi '\b[A-Z0-9._%-]+@[A-Z0-9.-]+@[A-Z]{2,4}' mail.txt
paul@hotmail.com
ringo@hotmail.com
alumno@alumno-VirtualBox:~$
```

Tanto el comando *sed* como el comando *grep*, pueden utilizarse sobre la salida de otro comando. Para poder hacer eso, debemos utilizar el carácter | y redirigir la salida del otro comando hacia *grep* o *sed*. Por ejemplo, podríamos utilizar el comando echo para imprimir un número, y esa salida reemplazarla por un string distinto.

```
alumno@alumno-VirtualBox:~$ echo "123" | sed 'y/[123]/[456]/'  
456
```

Estructura de directorios, archivos y puntos de montaje

***Arquitectura y Sistemas Operativos.
Tecnicatura Superior en Programación.
UTN-FRA***

Autores: Prof. Martín Isusi Seff

Revisores: Prof. Marcos Pablo Russo

Versión: 1



Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

Estructura de directorios

La estructura de directorios en GNU/Linux está ordenada siguiendo el estándar FHS (Estándar de jerarquía de archivos o Filesystem Hierarchy Standard), creado y mantenido por la organización Free Standards Group (Conformada por compañías de software y hardware como AMD, Debian, Dell, Google, HP, IBM, Intel, etc.). Este estándar define las bases para que tanto los programas del sistema, como los usuarios y administradores, sepan dónde encontrar lo que buscan. La mayoría de las distribuciones de GNU/Linux, inclusive las que forman parte de Free Software Standards, no aplican de forma estricta y al 100% el estándar, aunque las diferencias son mínimas. Los aspectos avanzados el estándar se pueden leer en el sitio <http://www.pathname.com/fhs/pub/fhs-2.3.html>.

Cuando hablamos del contenido de los directorios, podemos definir dos clasificaciones principales: estáticos/dinámicos o compartidos/restringidos

- Estáticos: Contienen binarios, bibliotecas, documentación y otros archivos que no cambian sin intervención del administrador. Pueden estar en dispositivos de sólo lectura y no es necesario que se hagan copias de seguridad frecuentemente.
- Dinámicos: Contienen archivos que no son estáticos. Deben encontrarse en dispositivos que sean de lectura-escritura. Para este tipo de directorios, sí es necesario que se realicen copias de seguridad frecuentes.
- Compartidos: Los directorios compartidos contienen archivos que pueden utilizarse en otro sistema que no sea el mismo que los almacena.
- Restringidos: Contiene archivos que no pueden compartirse.

A continuación, algunos ejemplos de directorios estáticos, dinámicos, compartidos y restringidos:

- Estáticos: **/bin, /sbin, /opt, /boot, /usr/bin**
- Dinámicos: **/var/mail, /var/spool, /var/run, /var/lock, /home**
- Compartidos: **/usr/bin, /opt**
- No compartidos: **/etc, /boot, /var/run, /var/lock**

Todos los archivos y directorios aparecen debajo del directorio raíz «/» (El equivalente en el mundo Unix al C:\ de Windows) aunque se encuentren en discos/dispositivos distintos. En Linux/Unix no existen letras de discos (C:, D:, etc.) Los dispositivos se 'montan' (empiezan a formar parte) del árbol de directorios del sistema.

A continuación. se listan los directorios más importantes y el contenido que almacenan:

Directorio	Contenido
/bin/	Comandos/programas binarios esenciales (cp, mv, ls, rm, etc.),
/boot/	Archivos utilizados durante el arranque del sistema (núcleo y discos RAM)
/dev/	Dispositivos esenciales, discos duros, terminales, sonido, video, lectores dvd/cd, etc

/etc/	Archivos de configuración utilizados en todo el sistema y que son específicos del ordenador
/etc/opt/	Archivos de configuración utilizados por programas alojados dentro de /opt/
/etc/X11/	Archivos de configuración para el sistema X Window (Opcional)
/etc/sgml/	Archivos de configuración para SGML (Opcional)
/etc/xml/	Archivos de configuración para XML (Opcional)
/home/	Directorios de inicios de los usuarios (Opcional)
/lib/	Bibliotecas compartidas esenciales para los binarios de /bin/, /sbin/ y el núcleo del sistema.
/mnt/	Sistemas de archivos montados temporalmente.
/media/	Puntos de montaje para dispositivos de medios como unidades lectoras de discos compactos.
/opt/	Paquetes de aplicaciones estáticas.
/proc/	Sistema de archivos virtual que documenta sucesos y estados del núcleo. Contiene principalmente archivos de texto.
/root/	Directorio de inicio del usuario root (super-usuario) (Opcional)
/sbin/	Comandos/programas binarios de administración de sistema.
/tmp/	Archivos temporales
/srv/	Datos específicos de sitio servidos por el sistema.
/usr/	Jerarquía secundaria para datos compartidos de solo lectura (Unix system resources). Este directorio puede ser compartido por múltiples ordenadores y no debe contener datos específicos del ordenador que los comparte.
/usr/bin/	Comandos/programas binarios.
/usr/include/	Archivos de inclusión estándar (cabeceras de cabecera utilizados para desarrollo).
/usr/lib/	Bibliotecas compartidas.
/usr/share/	Datos compartidos independientes de la arquitectura del sistema. Imágenes, archivos de texto, etc.
/usr/src/	Códigos fuente (Opcional)
/usr/X11R6/	Sistema X Window, versión 11, lanzamiento 6 (Opcional)
/usr/local/	Jerarquía terciaria para datos compartidos de solo lectura específicos del ordenador que los comparte.
/var/	Archivos variables, como son logs, bases de datos, directorio raíz de servidores HTTP y FTP, colas de correo, archivos temporales, etc.
/var/cache/	Cache da datos de aplicaciones.
/var/crash/	Depósito de información referente a caídas del sistema (Opcional)

/var/games/	Datos variables de aplicaciones para juegos (Opcional)
/var/lib/	Información de estado variable. Algunos servidores como MySQL y PostgreSQL almacenan sus bases de datos en directorios subordinados de éste.
/var/lock/	Archivos de bloqueo.
/var/log/	Archivos y directorios de registro del sistema (logs).
/var/mail/	Buzones de correo de usuarios (Opcional)
/var/opt/	Datos, variables de /opt/.
/var/spool/	Colas de datos de aplicaciones.
/var/tmp/	Archivos temporales preservados entre reinicios.

Archivos en Linux

Tanto GNU/Linux, como cualquier otro sistema operativo basado en Unix, es un sistema operativo completamente orientado a archivos. Esto quiere decir que todo (o casi todo) se representa con un archivo, tanto los datos (archivos de datos de cualquier tipo, como una canción o un programa) como los periféricos (mouse, teclado, placa de video, etc) o incluso los medios de comunicación (sockets, tuberías o *pipes*, etc.). Dado que tanto elementos de hardware, como datos, como medios de comunicación se representan como archivos, existen distintos tipos de archivos para cada caso: **los archivos directorio, los archivos comunes y los especiales**.

Archivos directorio

Los archivos directorios son una instancia especial de los archivos normales. Los directorios listan las localizaciones de otros archivos, algunos de los cuales pueden ser otros directorios.

Archivos comunes

Son archivos que pueden contener cualquier tipo de dato: Texto, Audio, Imagen, Scripts, Librerías de programación, etc.

Por defecto, nada permite diferenciar unos de otros, esto quiere decir, por ejemplo, que Linux no conoce la noción de extensión de archivo (aspecto que sí sirve para diferenciar un tipo de archivo de otro en Windows) como componente interno de la estructura del sistema de archivos, por lo que la extensión no tiene importancia y se considera simplemente parte del nombre.

Archivos especiales

Existen varios tipos de archivos especiales, pero principalmente sirven de interfaz para los diversos periféricos. Encontramos estos archivos especiales en el directorio /dev.

Nombre de archivos

Es necesario seguir reglas a la hora de nombrar archivos. Estas reglas son válidas para los tres tipos de archivos. Actualmente se puede llegar hasta 255 caracteres. Es muy importante tener en cuenta que Linux distingue entre mayúsculas y minúsculas, de modo que si tenemos un archivo nombrado "prueba", este archivo es distinto a un archivo nombrado "Prueba". Distinto sucede en, por ejemplo, en el sistema operativo Windows en cualquiera de sus versiones, donde no se diferencian mayúsculas y minúsculas y en el ejemplo antes mencionado, ambos archivos serían el mismo.

La mayoría de los caracteres (cifras, letras, ciertos signos, caracteres acentuados) son aceptados, incluyendo el espacio. A pesar de esto es necesario evitar ciertos caracteres reservados en la shell como son : & ; () ~ / \ ` ? - (al principio del nombre) <espacio>.

Rutas de archivos

Las rutas permiten definir la ubicación de un archivo en el sistema de archivos. Es la lista de directorios y sub-directorios utilizados para acceder a un sitio determinado de la estructura hasta la posición deseada (directorio, fichero).

El nombre de la ruta o path de un archivo es la concatenación, desde la raíz, de todos los directorios que se deben cruzar para acceder a él, que están separados cada uno por el carácter /.

Ejemplo: /home/usuario1/Documentos/Imagen.jpg

El ejemplo representa la ruta absoluta donde se ubica el archivo "Imagen.jpg". La ruta absoluta es aquella que se escribe desde el directorio raíz (/).

Además de las rutas absolutas, existen también las rutas relativas. Éstas se basan en la posición actual en el sistema de archivos. Siguiendo el ejemplo mostrado anterior, si utilizando la terminal nos ubicamos en el directorio "**usuario1**", la ruta relativa al archivo "**Imagen.jpg**" será **Documentos/Imagen.jpg**.

Para las rutas relativas se puede usar tanto el . (indica el directorio actual en el que estamos) como .. (indica el directorio superior).

Inodos y enlaces

En Linux, cada archivo en el sistema está representado por un inodo. Un inodo no es más que un bloque que almacena información de los archivos, de esta manera a cada inodo podemos asociarle un nombre. A simple vista pareciera que a un mismo archivo no podemos asociarle varios nombres, pero gracias a los enlaces esto es posible. Existen dos tipos de enlaces, los **enlaces físicos o duros** y los **enlaces simbólicos**.

Enlaces físicos o duros

Un enlace físico no es más que una etiqueta o un nuevo nombre asociado a un archivo. Es una

forma de identificar el mismo contenido con diferentes nombres. Éste enlace no es una copia separada del archivo anterior sino un nombre diferente para exactamente el mismo contenido. Para crear un enlace físico en Linux del archivo archivo.txt a nuevo_nombre.txt, ejecutamos el comando ln:

```
In archivo.txt nuevo_nombre.txt
```

El enlace aparecerá como otro archivo más en el directorio y apuntará al mismo contenido de archivo.txt. Cualquier cambio que se haga se reflejará de la misma manera tanto para archivo.txt como para nuevo_nombre.txt.

Un enlace se puede borrar usando el comando rm de la misma manera en que se borra un archivo, sin embargo el contenido del inodo no se eliminará mientras haya un enlace físico que le haga referencia. Esto puede tener varias ventajas, pero también puede complicar la tarea de seguimiento de los archivos. Un enlace físico no puede usarse para hacer referencia a directorios o a archivos en otros equipos.

Enlaces simbólicos

Un enlace simbólico también puede definirse como una etiqueta o un nuevo nombre asociado a un archivo pero a diferencia de los enlaces físicos, el enlace simbólico no contiene los datos del archivo, simplemente apunta al registro del sistema de archivos donde se encuentran los datos.

Tiene mucha similitud a un acceso directo en Windows o un alias en OS X.

Para crear un enlace simbólico del archivo archivo.txt a nuevo_nombre.txt, ejecutamos:

```
In -s archivo.txt nuevo_nombre.txt
```

Este enlace también aparecerá como otro archivo más en el directorio y apuntará al mismo contenido de archivo.txt, reflejando todos los cambios que se hagan tanto para archivo.txt como para nuevo_nombre.txt.

Sobre un enlace simbólico también se pueden usar todos los comandos básicos de archivos (rm, mv, cp, etc). sin embargo cuando el archivo original es borrado o movido a una ubicación diferente el enlace dejará de funcionar y se dice que el enlace está roto.

Un enlace simbólico permite enlazar directorios y también permite enlazar archivos fuera del equipo.

Puntos de montaje

La estructura de los sistemas de archivos está generalmente dividida en particiones, unidas todas ellas en el punto de montaje raíz (/) o separadas . Los sistemas de archivos de los dispositivos removibles como un USB o un Disco CD se unen a la raíz del sistema de la misma manera, como directorios o puntos de montaje. En principio estos directorios destinados a los dispositivos están vacíos, a la espera de su montaje, puede darse el caso de que el directorio destinado a este fin contenga subdirectorios o archivos, en cuyo caso quedarán ocultos hasta que el dispositivo se desmonte.

Para que las diferentes particiones estén disponibles desde un primer momento es necesario montarlas durante el arranque del sistema, los dispositivos removibles también se usan frecuentemente y es aconsejable tenerlos preparados para usar los comandos de montaje. Toda esta información se guarda en el archivo **/etc/fstab**. Los sistemas de archivos definidos en este fichero son revisados y montados durante el arranque del sistema. Sus entradas se consultan como fuente de información por defecto cuando los usuarios quieren montar dispositivos removibles.

Comando **mount** y **umount**

Después del arranque se pueden añadir más sistemas de archivos manualmente con el comando **mount**. El comando **mount** se usa para montar sistemas de archivos dentro de la estructura del árbol del sistema. El comando **mount** admite dos tipos de opciones, unos para el comando en sí, y otros para especificar opciones del sistema de ficheros.

mount [opciones] [dispositivo|directorio]

Las opciones que se pueden utilizar con el comando **mount** son:

-a	Lee el archivo /etc/fstab y monta todos los filesystem menos los que tengan la opción noauto
-h	Ayuda del comando
-o	Especifica las opciones del mount en la línea de comandos.
-r	Monta el filesystem en modo de solo lectura.
-t sftype	Especifica un tipo de filesystem.
-v	Salida interactiva.
-w	Monta el filesystem en modo de lectura/escritura.

Las opciones que pueden utilizarse con el comando **mount** y la opción **-o** son:

async	Toda la E/S al sistema de ficheros debería hacerse asincrónicamente.
auto	Puede montarse con la opción -a
defaults	Establece las opciones: rw, uid, dev, exec, auto, nouser y async . (todas las opciones por defecto).
dev	Interpretar dispositivos especiales de caracteres o bloques en el sistema de archivos.
exec	Permite sólo la ejecución de binarios.
noauto	Sólo puede montarse explícitamente (esto significa que la opción -a no hará que el filesystem se monte automáticamente).
noexec	No permite la ejecución de ningún binario en el sistema de archivos montado. Esta opción puede ser útil para servidores que tienen sistemas de archivos que contienen binarios para otras arquitecturas distintas.
nouserid	No permitir el efecto de los bits bits SUID ni SGID .

ro	Monta el filesystem en modo de sólo lectura.
rw	Monta el filesystem en modo de sólo escritura.
suid	Permite el efecto de los bits bits SUID ni SGID .
sync	Toda la E/S al filesystem debe hacerse sincrónicamente.
user	Permite a un usuario ordinario montar el filesystem.
users	Permite a cualquier usuario el montaje/desmontaje del filesystem.
remount	Vuelve a montar un filesystem ya montado.

Empaquetado de archivos y directorios utilizando el comando tar

El comando **tar** es utilizado normalmente para empaquetar o desempaquetar archivos. La sintaxis del comando es:

tar [parámetros] [archivo1] [archivo2]

Las opciones que se pueden usar con el comando **tar** son:

c	Crear un archivo.
v	Muestra la salida por pantalla.
x	Extrae los archivos.
z	Comprime/descomprime tar utilizando gzip (la extensión del empaqueado será .tar.gz o .tgz).
j	Comprime/descomprime tar utilizando bzip (la extensión del empaqueado será .tar.bz2 o .tbz2 o .tbz).
f	Imprime el menú con las opciones.
t	Muestra el contenido de un paquete.

Repositorios y paquetes

***Arquitectura y Sistemas Operativos.
Tecnicatura Superior en Programación.
UTN-FRA***

Autores: Prof. Martín Isusi Seff

Revisores: Prof. Marcos Pablo Russo

Versión: 1



Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

¿Qué es un repositorio?

Un repositorio es una "bodega" donde se encuentran los paquetes a instalar en cualquier distribución. Por lo general, los repositorios son servidores ftp o http (aunque también pueden ser locales) en donde se encuentran todos los paquetes disponibles para una distribución. Se habla de "paquetes" en vez de "programas", porque un paquete no necesariamente contiene un programa. Un paquete puede contener imágenes, librerías, código fuente, documentación, traducciones y desde luego programas. Cada distribución tiene sus propios repositorios y su forma de clasificarlos, y básicamente funcionan muy parecido. Generalmente, existe un programa en consola o en modo gráfico que se encarga de descargar y de instalar los paquetes, todo de modo automático.

Tipos de paquetes

Existen, por lo general, tres tipos de paquetes: estable, testing, inestable.

- Paquetes stable o estables: Consta de software estable y bien probado, y cambia sólo al incorporar correcciones importantes de seguridad.
- Paquetes en prueba o testing: Esta área contiene paquetes que se espera que sean parte de la próxima distribución estable. No goza de actualizaciones del equipo de seguridad en el mismo momento en que salen.
- Inestables o unstable: Contiene los paquetes más recientes. Una vez que un paquete ha cumplido las exigencias de estabilidad y calidad de empaquetado, será incluido en testing. Unstable tampoco está soportada por el equipo de seguridad.

Gestor de paquetes APT

APT (Adavance Packaging Tool) es el gestor de paquetes que se utiliza en distribuciones derivadas de Debian GNU/Linux: Crunchbang, Trisquel Ubuntu, Lubuntu, Linux Mint, etc... Es un sistema muy estable y muy fácil de usar a la vez, aquí se mostrarán los comandos principales para tener el control de la biblioteca **apt** de forma básica.

1) Actualizar la base de datos de paquetes.

```
# apt-get update
```

o

```
# apt update
```

Estos comandos leen el contenido del directorio **/etc/apt/sources.list** y **/etc/apt/sources.list.d/***, y actualizan la base de datos que se encuentra en el directorio **/var/lib/apt/lists/**.

2) Validar el estado de los paquetes

```
# apt-get check
```

Este comando chequea que la información que se ingresó en **sources.list** es correcta.

3) Actualizar paquetes

apt-get upgrade

Este comando actualiza los paquetes, es decir, verifica la versión de los paquetes instalados, y si hay nuevas versiones o dependencias, las actualiza. Previamente se debe ejecutar el comando **apt-get update**.

4) Actualizar la distribución

apt-get dist-upgrade

Esto nos permite actualizar la distribución, es decir una nueva actualización contiene nuevos paquetes como también software propio de la distribución que estamos utilizando.

5) Instalación de un paquete

apt-get install mc

El comando **apt-get install** instalará el paquete especificado. En el ejemplo, el paquete **mc** (Norton Commander)

6) Reinstalar un paquete

apt-get --reinstall mc

Esto es, si ya está instalado el paquete, y por alguna razón tiene problemas, este comando lo reinstalará.

7) Resolver dependencias

apt-get -f install

Este comando resuelve las dependencias de un paquete que hayamos instalado mediante el comando **dpkg**, es decir bajamos un paquete como puede ser virtualbox*.deb y desde la maquina lo instalamos. Esto necesita dependencias, y la forma de resolverlo es mediante este comando.

8) Remover paquetes

apt-get remove mc

Esto lo que hace es solo remover el paquete, si queremos remover los archivos de configuración también, ejecutamos el siguiente comando:

apt-get --purge remove mc

9) Limpiar los paquetes bajados

apt-get clean

Este comando borra el contenido del directorio **/var/cache/apt/archives/** que contiene todos los paquetes que bajamos de los repositorios. Antes de borrarlo, podemos copiarlo para que se instale en otro equipo.

10) Limpiar paquetes no usados

apt-get autoclean

Hay paquetes que se vuelven viejos y no se usan debido, por ejemplo, actualizaciones. En estos casos, este comando ayuda a removerlos.

11) Eliminar librerías innecesarias

apt-get autoremove

Cuando instalamos un paquete posiblemente se instalen dependencias junto con él. Si después borramos ese programa, quedan dependencias inservibles. Estas dependencias pueden borrarse con el anterior comando.

12) Buscar paquetes

apt-cache search paquete

Este comando busca paquetes en la base de datos que tenemos en el directorio **/var/lib/apt/lists/**.

13) Obtener sobre un paquete instalado la versión y el candidato

apt-cache policy paquetes

Con este comando obtenemos información sobre el paquete que tenemos instalado y cuál es el candidato a ser instalado.

Ej:

apt-cache policy mc

mc:

Instalados: 3:4.8.18-1

Candidato: 3:4.8.18-1

Tabla de versión:

***** 3:4.8.18-1 500**

500 http://ar.archive.ubuntu.com/ubuntu zesty/universe amd64

Packages

100 /var/lib/dpkg/status

14) Obtener información de un paquete instalado

apt-cache showpkg paquetes

La información que se muestra, se extrae de la base de datos de paquetes que se encuentra en el directorio **/var/lib/apt/lists/**.

15) Instalar un paquete que tengamos en nuestra pc (un archivo .deb)

```
# dpkg -i virtualbox-5.1_5.1.26-117224~Ubuntu~zesty_amd64.deb
```

Instala directamente el paquete que bajamos. A diferencia del comando **apt-get install paquete**, al usar el comando **dpkg -i**, no baja las dependencias necesarias. Por dicha razón, tenemos que usar el comando **apt-get -f install**.

16) Listar los paquetes instalados

```
# dpkg -l | more
```

Con este comando podemos ver todos los paquetes que tenemos instalados en nuestro equipo tanto sea por el comando apt-get install como también dpkg -i.

17) Ver contenido de los paquetes

```
# dpkg -L mc | more
```

Esto nos muestra todo el contenido del paquete que fue instalado, toda su estructura de archivos.

18) Borrar un paquete

```
# dpkg -r mc
```

Borrar un paquete instalado mediante dpkg -i.

19) Buscar a qué paquete pertenece un comando.

```
# which ls
```

```
# dpkg -S /bin/ls
```

20) Configurar un paquete

```
# dpkg --configure paquete
```

21) Reconfigurar todo

```
# dpkg --configure -a
```

Esto nos permite reconfigurar los paquetes que quedaron sin configurar.

Memorias

***Arquitectura y Sistemas Operativos.
Tecnicatura Superior en Programación.
UTN-FRA***

Autores: Prof. Martín Isusi Seff

Revisores: Prof. Marcos Pablo Russo

Versión: 1



Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

John Von Neumann, matemático y físico austriaco definió muchos de los principios que hoy rigen en el diseño de las computadoras. Uno de esos principios expresa que un programa debe estar almacenado en una memoria para poder ser ejecutado por la computadora.

Las computadoras tienden a acceder al almacenamiento en formas particulares. De hecho, la mayoría del acceso a almacenamiento tiende a exhibir uno (o ambos) de los siguientes atributos:

- El acceso tiende a ser secuencial
- El acceso tiende a ser localizado

El acceso secuencial significa que, si el CPU accede a la dirección N, es muy probable que la dirección N+1 sea la próxima a acceder. Esto tiene sentido, ya que muchos programas consisten de grandes secciones de instrucciones que ejecutan — en orden — una instrucción tras la otra.

El acceso localizado significa que, si se accede a la dirección X, es muy probable que otras direcciones alrededor de X también serán accedidas en el futuro.

Estos atributos son cruciales, debido a que permite que unidades de almacenamiento pequeña y más rápida, coloque efectivamente en memoria temporal almacenamiento más grande y lento. Esto es lo básico para implementar la memoria virtual. Pero antes de que discutamos la memoria virtual, debemos examinar las diferentes tecnologías de almacenamiento usadas actualmente.

Distintos tipos de memorias

Las computadoras de hoy utilizan una variedad de tecnologías de almacenamiento. Cada tecnología está orientada hacia una función específica, con velocidades y capacidades en combinación. Estas tecnologías son:

- Registros de CPU
- Memoria caché
- RAM
- Discos duros
- Almacenamiento fuera de línea para respaldos (cintas, discos ópticos, etc.)

En términos de capacidades y costos, estas tecnologías forman un espectro. Por ejemplo, los registros de CPU son:

- Muy rápidos (tiempos de acceso de unos pocos nanosegundos)
- Baja capacidad (usualmente menos de 200 bytes)
- Capacidades de expansión muy limitadas (se requiere un cambio en la arquitectura del CPU)
- Costosas

Sin embargo, en el otro lado del espectro, el almacenamiento fuera de línea es:

- Muy lento (tiempos de acceso se miden en días, si la media de respaldo debe ser entregada sobre largas distancias)
- Capacidad muy alta (10s - 100s de gigabytes)
- Capacidades de expansión prácticamente ilimitadas (solamente limitadas por el espacio físico requerido para hospedar la media de respaldo)
- Bajo costo

Usando diferentes tecnologías con diferentes capacidades, es posible afinar el diseño del sistema para un máximo rendimiento al costo más bajo posible. Las secciones siguientes exploran cada tecnología en el espectro del almacenamiento.

Registros de la CPU

Todos los diseños de CPU de hoy día incluyen registros para una variedad de propósitos, desde el almacenamiento de direcciones de la instrucción recientemente ejecutada, hasta propósitos más generales de almacenamiento y manipulación de datos. Los registros de CPU se ejecutan a la misma velocidad que el resto del CPU; de lo contrario habría un cuello de botella grave sobre el rendimiento completo del sistema. La razón para esto es que casi todas las operaciones realizadas por el CPU envuelven registros de una forma u otra.

El número de registros de CPU (y sus usos) dependen estrictamente en el diseño arquitectónico del CPU mismo. No hay forma de cambiar el número de registros de CPU, solamente puede migrar a un CPU con una arquitectura diferente. Por estas razones, el número de registros de CPU se puede considerar como una constante, ya que sólo pueden cambiarse con mucho dolor y grandes costos.

Memoria caché

El propósito de la memoria caché es actuar como una memoria temporal entre los registros de CPU, limitados y de gran velocidad y el sistema de memoria principal, mucho más grande y lento — usualmente conocido como RAM¹. La memoria caché tiene una velocidad de operación similar a la del CPU mismo, por eso cuando el CPU accede a datos en la caché, no tiene que quedarse esperando por los datos.

La memoria caché es configurada de forma tal que, cuando se leen datos desde la RAM, el sistema de hardware verifica primero para determinar si los datos deseados están en caché. Si los datos están en caché, estos son recuperados rápidamente y utilizados por el CPU. Sin embargo, si los datos no están en caché, estos se leen desde la RAM y, mientras se transfieren

¹ Mientras que "RAM" es un acrónimo para "Random Access Memory," y un término que podría ser fácilmente aplicable a cualquier tecnología de almacenamiento que permita el acceso no secuencial de los datos almacenados, cuando los administradores de sistemas hablan sobre RAM invariablemente se refieren al sistema de memoria principal.

al CPU, también se colocan en caché (en caso de que se necesiten más tarde). Desde la perspectiva del CPU, todo esto se hace de forma transparente, por lo que la única diferencia entre el acceso de los datos en caché y desde la RAM es la cantidad de tiempo que toma para que los datos sean recuperados.

En términos de la capacidad de almacenamiento, la caché es mucho más pequeña que la RAM. Por lo tanto, no todos los bytes en la RAM tienen su ubicación única en caché. Como tal, es necesario dividir la caché en secciones que se puedan utilizar para alojar diferentes áreas de RAM y tener un mecanismo que permita que cada área de la caché haga un "caché" de la RAM en diferentes momentos. Aunque existe una diferencia en tamaño entre la caché y la RAM, dada la naturaleza secuencial y localizada del acceso a almacenamiento, una pequeña cantidad de caché puede efectivamente acelerar el acceso a grandes cantidades de RAM.

Cuando se escriben datos desde el CPU, las cosas se complican un poco. Existen dos enfoques que se pueden utilizar. En ambos casos, los datos son escritos primero a la caché. Sin embargo, puesto que el propósito de la caché es funcionar como una copia muy rápida de los contenidos de porciones seleccionadas de RAM, cada vez que una porción de datos cambia su valor, ese nuevo valor debe ser escrito tanto a la caché como a la RAM. De lo contrario, los datos en caché y los datos en la RAM ya no coincidirían.

Los dos enfoques se diferencian en cómo se logra hacer esto. En un enfoque, conocido como write-through caching, los datos modificados se escriben inmediatamente a la RAM. Sin embargo, en el write-back caching, se retrasa la escritura de los datos modificados a la RAM. La razón para hacer esto es la de reducir el número de veces que una porción de datos modificada frecuentemente debe ser escrita nuevamente a la RAM.

La caché "write-through" o inmediata es un poco más simple de implementar; por esta razón es la más común. La caché "write-back" es un poco más complicada; además de almacenar los datos, es necesario mantener cierto tipo de mecanismo que sea capaz de notificar que los datos en caché están al día o "limpios" (los datos en caché son los mismos que los datos en RAM), o que están "sucios" (los datos en caché han sido modificados, lo que significa que los datos en RAM ya no están actualizados). También es necesario implementar una forma de vaciar periódicamente entradas "sucias" en caché de vuelta en RAM.

Niveles de caché

Los subsistemas de caché en los diseños de computadoras de hoy día pueden ser de niveles múltiples; esto es, puede haber más de un conjunto de caché entre el CPU y la memoria principal. Los niveles de caché a menudo están enumerados, con los números menores más cercanos a la CPU. Muchos sistemas tienen dos niveles de caché:

- La caché L1 a menudo está ubicada en el chip del CPU mismo y se ejecuta a la misma velocidad que el CPU.

- La caché L2 usualmente es parte del módulo de CPU, se ejecuta a las mismas velocidades que el CPU (o casi) y normalmente es un poco más grande y lenta que la caché L1

Algunos sistemas (normalmente servidores de alto rendimiento) también tienen caché L3, que usualmente forma parte del sistema de la tarjeta madre. Como puede imaginarse, la caché L3 es más grande (y casi con seguridad más lenta) que la caché L2.

En cualquier caso, el objetivo de todos los subsistemas de caché — bien sean simples o de múltiples niveles — es el de reducir el tiempo de acceso promedio a la RAM.

Memoria principal – RAM

La RAM resuelve la mayoría del almacenamiento electrónico en las computadoras de hoy en día. La RAM es utilizada tanto para almacenar datos como para almacenar los programas en uso. La velocidad de la RAM en la mayoría de los sistemas actuales está entre la velocidad de la memoria caché y la de los discos duros y está mucho más cercana a la velocidad de la primera que a la segunda.

La operación básica de la RAM es en realidad bien sencilla. En el nivel más bajo, están los chips de RAM — circuitos integrados que "recuerdan". Estos chips tienen cuatro tipos de conexiones con el mundo externo:

- Conexiones de energía (para operar la circuitería dentro del chip)
- Conexiones de datos (para permitir la transferencia de datos hacia adentro y fuera del chip)
- Conexiones de lectura/escritura (para controlar si los datos se almacenaran o se recuperaran desde el chip)
- Conexiones de direcciones (para determinar si los datos en el chip serán leídos/escritos)

He aquí los pasos requeridos para almacenar datos en RAM:

- Los datos a almacenar se presentan a las conexiones de datos.
- La dirección en la que los datos se almacenaran se presenta a las conexiones de dirección.
- La conexión de lectura/escritura se coloca en modo de escritura.

La recuperación de datos es también muy directa:

- La dirección de los datos deseados se presenta a las conexiones de direcciones.
- La conexión de lectura/escritura es colocada a modo de lectura.
- Los datos deseados son leídos desde las conexiones de datos.

Mientras que estos pasos parecen bastante simples, estos toman lugar a grandes velocidades, con el tiempo consumido en cada paso medido en nanosegundos.

Casi todos los chips de memoria creados hoy en día se venden como módulos. Cada módulo consiste de un número individual de chips de RAM conectados a una pequeña tarjeta de circuitos. La distribución mecánica y eléctrica del módulo sigue los estándares de la industria,

haciendo posible la compra de memorias de sde diferentes fabricantes.

Discos duros

Todas las tecnologías discutidas hasta ahora son volátiles por naturaleza. En otras palabras, los datos contenidos en almacenamiento volátil se pierden cuando se desconecta el poder.

Por otro lado, los discos duros son no-volátiles — lo que significa que los datos se mantienen allí, aún después que se ha desconectado la energía. Debido a esto, los discos duros ocupan un lugar muy especial en el espectro del almacenamiento. Su naturaleza no-volátil los hace ideal para el almacenamiento de programas y datos para su uso a largo plazo. Otro aspecto único de los discos duros es que, a diferencia de la RAM y la memoria caché, no es posible ejecutar los programas directamente cuando son almacenados en discos duros; en vez de esto, ellos deben primero ser leídos a la RAM.

Otra diferencia de la caché y de la RAM es la velocidad del almacenamiento y recuperación de los datos; los discos duros son de al menos un orden de magnitud más lento que todas las tecnologías electrónicas utilizadas para caché y RAM. La diferencia en velocidad es debida principalmente a su naturaleza electromecánica. Hay cuatro fases distintas que toman lugar durante cada transferencia de datos desde o hacia un disco duro. La lista siguiente ilustra estas fases, junto con el tiempo que en promedio tomaría una unidad de alto rendimiento, para completar cada fase:

- Movimiento del brazo de acceso (5.5 milisegundos)
- Rotación del disco (.1 milisegundos)
- Lectura/escritura de datos de cabezales (.00014 milisegundos)
- Transferencia de datos hacia/desde la electrónica del disco (.003 Milisegundos)

Fundamentos de la memoria virtual

Aun cuando la tecnología detrás de la construcción de las implementaciones modernas de almacenamiento es realmente impresionante, el administrador de sistemas promedio no necesita estar al tanto de los detalles. De hecho, solamente existe un factor que los administradores de sistemas deberían tener en consideración: Nunca hay suficiente RAM.

Mientras que esta frase puede sonar al principio un poco cómica, muchos diseñadores de sistemas operativos han empleado una gran cantidad de tiempo tratando de reducir el impacto de esta limitación. Esto lo han logrado mediante la implementación de la memoria virtual — una forma de combinar RAM con un almacenamiento más lento para darle al sistema la apariencia de tener más RAM de la que tiene instalada realmente.

Memoria virtual en términos sencillos

Vamos a comenzar con una aplicación hipotética. El código de máquina que conforma esta

aplicación tiene un tamaño de 10000 bytes. También requiere otros 5000 bytes para el almacenamiento de datos y para la memoria intermedia de E/S. Esto significa que, para ejecutar la aplicación, deben haber más de 15000 bytes de RAM disponible; un byte menos y la aplicación no será capaz de ejecutarse.

Este requerimiento de 15000 bytes se conoce como el espacio de direcciones de la aplicación. Es el número de direcciones únicas necesarias para almacenar la aplicación y sus datos. En las primeras computadoras, la cantidad de RAM disponible tenía que ser mayor que el espacio de direcciones de la aplicación más grande a ejecutar; de lo contrario, la aplicación fallaría con un error de "memoria insuficiente".

Un enfoque posterior conocido como solapamiento intentó aliviar el problema permitiendo a los programadores dictar cuales partes de sus aplicaciones necesitaban estar residentes en memoria en cualquier momento dado. De esta forma, el código requerido solamente para propósitos de inicialización podía ser sobrescrito con código a utilizar posteriormente. Mientras que el solapamiento facilitó las limitaciones de memoria, era un proceso muy complejo y susceptible a errores. El solapamiento también fallaba en solucionar el problema de las limitaciones de memoria globales al sistema en tiempo de ejecución. En otras palabras, un programa con solapamiento requería menos memoria para ejecutarse que un programa sin solapamiento, pero si el sistema no tiene suficiente memoria para el programa solapado, el resultado final es el mismo — un error de falla de memoria.

Con la memoria virtual el concepto del espacio de direcciones de las aplicaciones toma un significado diferente. En vez de concentrarse en cuanta memoria necesita una aplicación para ejecutarse, un sistema operativo con memoria virtual continuamente trata de encontrar una respuesta a la pregunta "¿qué tan poca memoria necesita la aplicación para ejecutarse?".

Aunque inicialmente pareciera que nuestra aplicación hipotética requiere de un total de 15000 bytes para ejecutarse — el acceso a memoria tiende a ser secuencial y localizado. Debido a esto, la cantidad de memoria requerida para ejecutar la aplicación en un momento dado es menos que 15000 bytes — usualmente mucho menos. Considere los tipos de accesos de memoria requeridos para ejecutar una instrucción de máquina sencilla:

- La instrucción es leída desde la memoria.
- Se leen desde memoria los datos requeridos por la instrucción.
- Despues de completar la instrucción, los resultados de la instrucción son escritos nuevamente en memoria.

El número real de bytes necesarios para cada acceso de memoria varían de acuerdo a la arquitectura del CPU, la instrucción misma y el tipo de dato. Sin embargo, aún si una instrucción requiere de 100 bytes de memoria por cada tipo de acceso de memoria, los 300 bytes requeridos son mucho menos que el espacio de direcciones total de la aplicación de

15000 bytes. Si hubiese una forma de hacer un seguimiento de los requerimientos de memoria de la aplicación a medida que esta se ejecuta, sería posible mantener la aplicación ejecutándose usando menos memoria que lo que indicaría su espacio de direcciones.

Pero esto genera una pregunta: Si solamente una parte de la aplicación está en memoria en un momento dado, ¿dónde está el resto?

La respuesta corta a esta pregunta es que el resto de la aplicación se mantiene en disco. En otras palabras, el disco actúa como un almacenamiento de respaldo para la RAM; un medio más lento y también más grande que actúa como un "respaldo" para un almacenamiento más rápido y más pequeño. Esto puede parecer al principio como un gran problema de rendimiento en su creación — después de todo, las unidades de disco son mucho más lentas que la RAM.

Aunque esto es cierto, es posible tomar ventaja del comportamiento de acceso secuencial y localizado de las aplicaciones y eliminar la mayoría de las implicaciones de rendimiento en el uso de unidades de disco como unidades de respaldo para la RAM. Esto se logra estructurando el subsistema de memoria virtual para que este trate de asegurarse de que esas partes de la aplicación que actualmente se necesitan — o que probablemente se necesitarán en un futuro cercano — se mantengan en RAM solamente por el tiempo en que son realmente requeridas.

En muchos aspectos, esto es similar a la situación entre la caché y la RAM: haciendo parecer una poca cantidad de almacenamiento rápido con grandes cantidades de un almacenamiento lento actuar como que si se tratase de grandes cantidades de almacenamiento rápido.

Con esto en mente, exploremos el proceso en más detalle.

Memoria virtual: Los detalles

Primero, debemos introducir un nuevo concepto: espacio de direcciones virtuales. El espacio de direcciones virtuales es el espacio de direcciones máximo disponible para una aplicación. El espacio de direcciones virtuales varía de acuerdo a la arquitectura del sistema y del sistema operativo. El espacio de direcciones virtuales depende de la arquitectura puesto que es la arquitectura la que define cuántos bits están disponibles para propósitos de direccionamiento. El espacio de direcciones virtuales también depende del sistema operativo puesto que la forma en que el sistema operativo fue implementado puede introducir límites adicionales sobre aquellos impuestos por la arquitectura.

La palabra "virtual" en el espacio de direcciones virtuales, significa que este es el número total de ubicaciones de memoria direccionables disponibles para una aplicación, pero no la cantidad de memoria física instalada en el sistema, o dedicada a la aplicación en un momento dado.

En el caso de nuestra aplicación de ejemplo, su espacio de direcciones virtuales es de 15000 bytes.

Para implementar la memoria virtual, para el sistema es necesario tener un hardware especial de administración de memoria. Este hardware a menudo se conoce como un MMU (Memory

Management Unit). Sin un MMU, cuando el CPU accede a la RAM, las ubicaciones reales de RAM nunca cambian — la dirección de memoria 123 siempre será la misma dirección física dentro de la RAM.

Sin embargo, con un MMU, las direcciones de memoria pasan a través de un paso de traducción antes de cada acceso de memoria. Esto significa que la dirección de memoria 123 puede ser redirigida a la dirección física 82043 en un momento dado y a la dirección 20468 en otro. Como resultado de esto, la sobrecarga relacionada con el seguimiento de las traducciones de memoria virtual a física sería demasiado. En vez de esto, la MMU divide la RAM en páginas — secciones contiguas de memoria de un tamaño fijo que son manejadas por el MMU como unidades sencillas.

Mantener un seguimiento de estas páginas y sus direcciones traducidas puede sonar como un paso adicional confuso e innecesario, pero de hecho es crucial para la implementación de la memoria virtual. Por tal razón, considere el punto siguiente:

Tomando nuestra aplicación hipotética con un espacio de direcciones virtuales de 15000 bytes, asuma que la primera instrucción de la aplicación accede a los datos almacenados en la dirección 12374. Sin embargo, también asuma que nuestra computadora solamente tiene 12288 bytes de RAM física. ¿Qué pasa cuando el CPU intenta acceder a la dirección 12374?

Lo que ocurre se conoce como un fallo de página.

Memoria secundaria. Conceptos fundamentales y administración

Anteriormente, en este capítulo se habló del concepto de memoria principal. Aquí se almacenan aquellos programas que se deben ser ejecutados por la computadora. Si bien la memoria virtual basa su funcionamiento en el disco rígido que es no volátil, la memoria principal sí es volátil. Esto no sucede en la memoria secundaria: La memoria secundaria es aquella memoria no volátil y que sirve de soporte al sistema, almacenando todos los programas de manera permanente, y desde la cual se toman los programas que deben ser ejecutados, para luego moverlos a la memoria principal.

Los discos rígidos son el dispositivo más utilizado como memoria secundaria, aunque hoy en día existen alternativas. La alternativa principal es el almacenamiento de estado sólido (conocido como "disco de estado sólido" o SSD, por sus siglas en inglés Solid State Disk). Si bien este no es un disco rígido, ya que no posee platos metálicos ni ningún tipo de parte mecánica, puede ser manejado como un disco rígido regular.

Una vez que el disco principal está instalado, poco se puede hacer con él sin antes establecer particiones y especificar un sistema de archivos.

¿Qué significa "particionar" un disco?

Partitionar un disco significa dividir un disco físico en varios discos lógicos. Una partición es un conjunto de bloques contiguos dentro de un disco rígido físico, y cada partición es interpretada

por el sistema operativo como discos independientes. El disco rígido, necesitará lo que se conoce como “tabla de particiones”, para poder especificar qué sectores físicos del disco corresponden a cada partición.

Muchas veces, conociendo el concepto de partición surge la pregunta ¿Por qué tener múltiples particiones?

- Tener múltiples particiones permite “encapsular” los datos. Esto quiere decir que cada partición tendrá su propio sistema de archivos, que será independiente del resto de las particiones. En caso de corrupción, la pérdida de datos se limita a esa partición.
- Múltiples particiones pueden tener formatos distintos, variando, por ejemplo, el tipo de sistema de archivos o la cantidad de bloques en la que se divide lógicamente el disco. Esto hace que podamos tener un mismo disco, particionado para distintos usos, lo que mejora el rendimiento del sistema.
- Realizando particiones podemos limitar el tamaño que puede utilizar un proceso o un usuario. Por ejemplo, podríamos realizar una partición en la que se almacene el sistema operativo y una partición para los datos del usuario. Si los datos del usuario completan el tamaño disponible en su partición, esto no hará que el sistema operativo se quede sin espacio para trabajar.

Existen distintos tipos de particiones:

- Las particiones primarias son particiones que toman hasta cuatro de las ranuras de particiones en la tabla de particiones del disco duro.
- Las particiones extendidas fueron desarrolladas en respuesta a la necesidad de más de cuatro particiones por unidad de disco. Una partición extendida puede contener dentro de sí múltiples particiones, extendiendo el número de particiones posibles en una sola unidad de disco. La introducción de las particiones extendidas se generó por el desarrollo constante de las capacidades de los discos duros.

Las particiones lógicas son aquellas que están contenidas dentro de una partición extendida; en términos de uso, son iguales a una partición primaria no extendida.

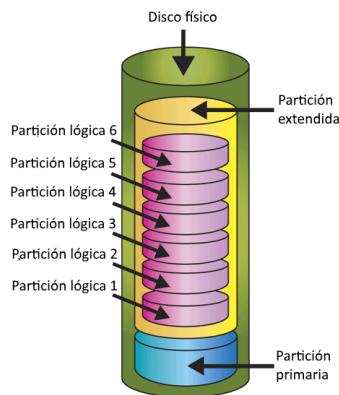


Figura 1 Diagrama representativo de los distintos tipos de particiones.

Sistema de archivos

Aun teniendo el dispositivo de almacenamiento configurado y particionado correctamente, sería difícil almacenar y recuperar información — todavía nos falta una forma de estructurar y organizar esa información. Lo que necesitamos es un sistema de archivos.

El concepto de un sistema de archivos es tan fundamental para el uso de los dispositivos de almacenamiento masivo que el usuario de computadoras promedio ni siquiera hace una distinción entre los dos. Sin embargo, los administradores de sistemas no se pueden permitir ignorar los sistemas de archivos y su impacto en el trabajo diario.

Un sistema de archivos es un método para representar datos en un dispositivo de almacenamiento masivo. Los sistemas de archivos usualmente incluyen las características siguientes:

- Almacenamiento de datos basado en archivos
- Estructura de directorio jerárquico (algunas veces llamado "carpeta")
- Seguimiento de la creación de archivos, tiempos de acceso y de modificación
- Algún nivel de control sobre el tipo de acceso permitido para un archivo específico
- Un concepto de propiedad de archivos
- Contabilidad del espacio utilizado

Almacenamiento de datos basado en archivos

Mientras que los sistemas de archivos que utilizan esta metáfora para el almacenamiento de datos son prácticamente universales que casi se consideran como la norma, todavía existen varios aspectos que se deben considerar.

Primero debe estar consciente de cualquier restricción de nombres. Por ejemplo, ¿cuáles son los caracteres permitidos en un nombre de archivo? ¿Cuál es el largo máximo para un nombre de archivo? Estas preguntas son importantes, pues dictan cuáles nombres de archivos se

pueden utilizar y cuáles no. Los sistemas operativos más antiguos con sistemas de archivos más primitivos permitían solamente caracteres alfanuméricicos (y solamente mayúsculas) y únicamente nombres de archivos 8.3 (lo que significa un nombre de archivo de ocho caracteres, seguido de una extensión de tres caracteres).

Estructura de directorios jerárquico

Mientras que los sistemas de archivos en ciertos sistemas operativos antiguos no incluían el concepto de directorios, todos los sistemas de archivos de hoy día incluyen esta característica. Los directorios son usualmente implementados como archivos, lo que significa que no se requiere de utilidades especiales para mantenerlos.

Más aún, puesto que los directorios son en sí mismos archivos, y los directorios contienen archivos, los directorios pueden a su vez contener otros directorios, conformando una estructura jerárquica de múltiples niveles. Este es un concepto poderoso con el cual muchos administradores de sistemas deberían de estar familiarizados. Usando las jerarquías de múltiples niveles puede hacer la administración de archivos mucho más fácil para usted y sus usuarios.

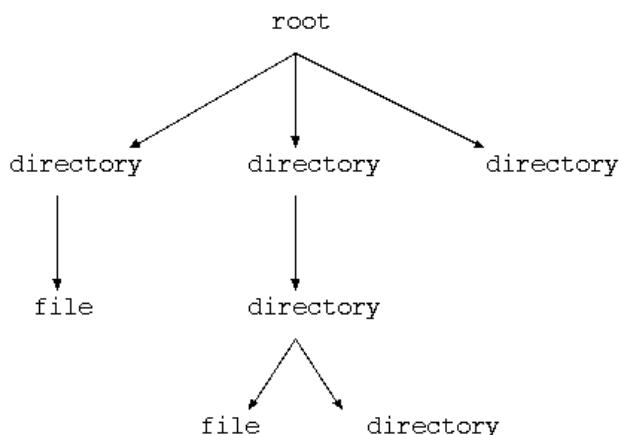


Figura 2 En una estructura de directorios jerárquicos, tendremos un directorio principal dentro del cual se ubicarán otros directorios o incluso archivos. Más adelante se verá la estructura de directorios utilizada en sistemas operativos GNU/Linux

Seguimiento de la creación de archivos, tiempo de acceso y modificación

La mayoría de los sistemas de archivos mantienen un seguimiento del tiempo en el que se creó un archivo; otros mantienen un seguimiento de los tiempos de acceso y modificación. Más allá de la conveniencia de poder determinar cuándo un archivo dado fue creado, accedido o modificado, estas fechas son vitales para la operación adecuada de los respaldos incrementales.

Control de acceso

El control de acceso es un área en la que los sistemas de archivos difieren dramáticamente. Algunos sistemas de archivos no tienen un modelo claro para el control de acceso, mientras que otros son mucho más sofisticados. En términos generales, la mayoría de los sistemas de

archivos modernos combinan dos componentes en una metodología cohesiva de control de acceso:

- Identificación del usuario
- lista de acciones permitidas

La identificación de usuarios significa que el sistema de archivos (y el sistema operativo subyacente) primeramente debe ser capaz de identificar únicamente a usuarios individuales. Esto hace posible tener una responsabilidad completa con respecto a cualquier operación a nivel de sistema de archivos. Otra funcionalidad de ayuda es la de los grupos de usuarios. Se utilizan grupos más a menudo en organizaciones donde los usuarios pueden ser miembros de uno o más proyectos. Otra funcionalidad que algunos sistemas de archivos soportan es la creación de identificadores genéricos que se pueden asignar a uno o más usuarios.

Luego, el sistema de archivos debe ser capaz de mantener listas de las acciones que son permitidas (o prohibidas) para cada archivo. Las acciones a las que se les hace seguimiento más a menudo son:

- Leer el archivo
- Escribir al archivo
- Ejecutar el archivo

Varios sistemas de archivos pueden extender la lista para incluir otras acciones tales como eliminar, o hasta la habilidad de hacer cambios al control de acceso del archivo.

Contabilidad del espacio utilizado

Una constante en la vida de un administrador de sistemas es la de que nunca hay suficiente espacio libre, y aún si lo hubiese, no estará disponible por mucho tiempo. Por lo tanto, un administrador de sistemas debería al menos ser capaz de determinar fácilmente el nivel de espacio libre disponible para cada sistema de archivos. Además, los sistemas de archivos con capacidades de identificación de usuarios bien definidas, a menudo incluyen la característica de mostrar la cantidad de espacio que un usuario particular ha consumido.

Esta característica es vital en grandes entornos de usuarios, pues es un hecho que la regla de 80/20 se aplica a menudo al espacio en disco — 20 por ciento de sus usuarios serán responsables por el consumo de 80 por ciento de su espacio disponible en disco. Al facilitar la identificación de estos usuarios en el 20 por ciento, puede más efectivamente manejar sus activos relacionados al almacenamiento.

Tomando este paso un poco más allá, algunos sistemas de archivos incluyen la habilidad de establecer los límites de uso del usuario (conocidos comúnmente como cuotas de disco) en la cantidad de espacio en disco que pueden consumir. Los detalles específicos varían de un sistema de archivos al otro, pero en general a cada usuario se le puede asignar una cantidad

específica de almacenamiento que un usuario puede utilizar. Más allá de allí, los sistemas de archivos varían. Algunos sistemas de archivos permiten que el usuario se exceda de su límite solamente una vez, mientras que otros implementan un "periodo de gracia" durante el que aplica un segundo límite más alto.

Tecnologías avanzadas de almacenamiento

Aunque todo lo que se ha presentado hasta ahora en este capítulo solamente trata con discos duros únicos conectados directamente a un sistema, existen otras opciones más avanzadas.

Almacenamiento accesible a través de la red

Combinando redes con las tecnologías de almacenamiento masivo puede resultar en una flexibilidad excelente para los administradores de sistemas. Con este tipo de configuración se tienen dos beneficios posibles:

- Consolidación del almacenamiento
- Administración simplificada

El almacenamiento se puede consolidar implementando servidores de alto rendimiento con conexiones de red de alta velocidad y configurados con grandes cantidades de almacenamiento rápido. Con la configuración apropiada, es posible suministrar acceso al almacenamiento a velocidades comparables al almacenamiento conectado directamente. Más aún, la naturaleza compartida de tal configuración a menudo hace posible reducir los costos, ya que los gastos asociados con suministrar almacenamiento centralizado y compartido pueden ser menores que suministrar el almacenamiento equivalente para cada uno de los clientes. Además, el espacio libre está consolidado, en vez de esparcido (pero no utilizable globalmente) entre los clientes.

Los servidores de almacenamiento centralizado también pueden hacer muchas tareas administrativas más fáciles. Por ejemplo, monitorizar el espacio libre es mucho más fácil cuando el almacenamiento a supervisar existe en un sólo servidor centralizado. Los respaldos también se pueden simplificar en gran medida usando un servidor de almacenamiento centralizado. Es posible hacer respaldos basados en la red para múltiples clientes, pero se requiere más trabajo para configurar y mantener.

Existen varias tecnologías disponibles de almacenamiento en red; seleccionar una puede ser complicado. Casi todos los sistemas operativos en el mercado hoy día incluyen alguna forma de acceder a almacenamiento en red, pero las diferentes tecnologías son incompatibles entre ellas.

Almacenamiento basado en RAID

Una habilidad que un administrador de sistemas debería cultivar es la de leer las complejas configuraciones de sistemas y observar las diferentes limitaciones inherentes a cada

configuración. Mientras que esto, a primera vista, puede parecer un punto de vista deprimente a tomar, puede ser una forma excelente de ver más allá de las brillantes cajas nuevas y visualizar una futura noche del sábado con toda la producción detenida debido a una falla que se podría haber evitado fácilmente con pensar un poco.

Con esto en mente, utilicemos lo que conocemos sobre el almacenamiento basado en discos y veamos si podemos determinar las formas en que los discos pueden causar problemas. Primero, considere una falla absoluta del hardware:

Un disco duro con cuatro particiones se muere completamente: ¿qué pasa con los datos en esas particiones?

Está indisponible inmediatamente (al menos hasta que la unidad dañada pueda ser reemplazada y los datos restaurados desde el respaldo más actual).

Un disco duro con una sola partición está operando en los límites de su diseño debido a cargas de E/S masivas: ¿qué les pasa a las aplicaciones que requieren acceso a los datos en esa partición?

Las aplicaciones se vuelven más lentas debido a que el disco duro no puede procesar lecturas y escrituras más rápido.

Tiene un archivo de datos que poco a poco sigue creciendo en tamaño; pronto será más grande que el disco más grande disponible en su sistema. ¿Qué pasa entonces?

La unidad de disco se llena, el archivo de datos deja de crecer y su aplicación asociada deja de funcionar.

Cualquiera de estos problemas puede dejar su centro de datos inútil, sin embargo, los administradores de sistemas deben enfrentarse a este tipo de problemas todos los días. ¿Qué se puede hacer?

Afortunadamente, existe una tecnología que puede resolver cada uno de estos problemas. El nombre de esta tecnología es RAID.

RAID es el acrónimo para Redundant Array of Independent Disks, Formación de Discos Independientes Redundantes². Como su nombre lo implica, RAID es una forma para que discos múltiples actúen como si se tratases de una sola unidad.

Las técnicas RAID primero fueron desarrolladas por investigadores en la Universidad de California, Berkeley a mitad de los 80. En ese tiempo, había una gran separación entre las unidades de disco de alto rendimiento utilizadas por las grandes instalaciones computacionales del momento, y las unidades de disco más pequeñas utilizadas por la joven industria de las computadoras personales. RAID se veía como el método para tener varios discos menos costosos sustituyendo una unidad más costosa.

² Cuando comenzaron las primeras investigaciones de RAID, el acrónimo venía de Redundant Array of Inexpensive Disks, pero con el paso del tiempo los discos "independientes" que RAID pretendía sustituir se volvieron más y más económicos, dejando la cuestión del costo un poco sin sentido.

Más aún, las formaciones RAID se pueden construir de varias formas, resultando en características diferentes dependiendo de la configuración final. Vamos a ver las diferentes configuraciones (conocidas como niveles RAID) en más detalles.

Niveles de RAID

Al principio, fueron definidos cinco niveles RAID y los nombraron del "1" al "5." Luego, otros investigadores y miembros de la industria del almacenamiento definieron niveles RAID adicionales. No todos los niveles RAID eran igualmente útiles; algunos eran de interés solamente para propósitos de investigación y otros no se podían implementar de una forma económica.

Al final, había tres niveles de RAID que terminaron siendo ampliamente utilizados:

- Nivel 0
- Nivel 1 (RAID I)
- Nivel 5 (RAID V)

RAID 0

Se utiliza para doblar el rendimiento y para fusionar todos los discos duros en un sólo disco para aumentar la capacidad de almacenamiento. Es necesario tener 2 discos duros como mínimo. Por ejemplo, si tenemos dos discos que funcionan a una velocidad alrededor de 20 Mb/s, al poner dos discos se duplicaría la velocidad es decir 40 Mb/s (2x20 Mb/s). Es una partición lógica cuyo tamaño es igual a la suma de los discos integrados en el sistema RAID.

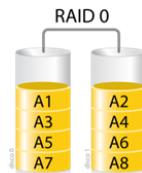


Figura 3

RAID 1

Es utilizado para garantizar la integridad de los datos, en caso de un fallo de uno de los discos duros, es posible continuar las operaciones en el otro disco duro sin ningún problema. No se mejora el rendimiento y no se suman el espacio de los discos como en RAID 0. El tipo de RAID 1 se llama comúnmente "mirroring" debido a que éste hace una simple copia del primer disco.

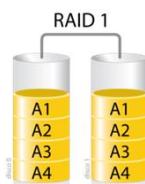


Figura 4

RAID 5

RAID 5 trata de combinar los beneficios de RAID 0 y RAID 1, a la vez que trata de minimizar sus desventajas. Igual que RAID 0, un RAID 5 consiste de múltiples unidades de disco, cada una dividida en porciones. Esto permite a una formación RAID 5 ser más grande que una unidad individual. Como en RAID 1, una formación RAID 5 utiliza algo de espacio en disco para alguna forma de redundancia, mejorando así la confiabilidad.

Una formación RAID 5 debe consistir de al menos tres discos idénticos en tamaño (aunque se pueden utilizar más discos). Cada unidad está dividida en porciones y los datos se escriben a las porciones siguiendo un orden. Sin embargo, no cada porción está dedicada al almacenamiento de datos como en RAID 0. En cambio, en una formación con n unidades en ella, la enésima porción está dedicada a la paridad.

Las porciones que contienen paridad hacen posible recuperar los datos si falla una de las unidades en la formación. La paridad en la porción x se calcula matemáticamente combinando los datos desde cada porción x almacenado en todas las otras unidades en la formación. Si los datos en una porción son actualizados, la correspondiente porción de paridad debe ser

recalculada y actualizada también.

Esto también significa que cada vez que se escriben datos en la formación, al menos dos unidades son escritas a: la unidad almacenando los datos y la unidad que contiene la porción con la paridad.

Un punto clave a tener en mente es que las porciones de paridad no están concentradas en una sola unidad de la formación. En cambio, están distribuidas uniformemente a lo largo de todas las unidades. Aun cuando es posible dedicar una unidad específica para que contenga únicamente paridad (de hecho, esta configuración se conoce como RAID nivel 4), la actualización constante de la paridad a medida que se escriben datos a la formación significa que la unidad de paridad se podría convertir en un cuello de botella. Distribuyendo la información de paridad uniformemente a través de la formación, se reduce este impacto.

Sin embargo, es importante recordar el impacto de la paridad en la capacidad general de almacenamiento de la formación. Aun cuando la información de paridad se distribuye uniformemente a lo largo de todos los discos, la cantidad de almacenamiento disponible se reduce por el tamaño de un disco.

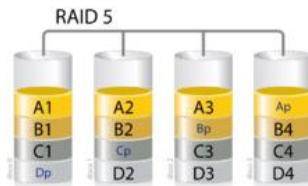


Figura 5

Niveles RAID anidados

Como debería ser obvio a partir de la discusión sobre los diferentes niveles de RAID, cada nivel tiene sus fortalezas y debilidades específicas. No fue mucho tiempo después de que se comenzó a implementar el almacenamiento basado en RAID que la gente comenzó a preguntarse si los diferentes niveles RAID se podrían combinar de alguna forma, produciendo formaciones con todas las fortalezas y ninguna de las debilidades de los niveles originales.

Por ejemplo, ¿qué pasa si los discos en una formación RAID 0 fuesen en verdad formaciones RAID 1? Esto proporcionaría las ventajas de la velocidad de RAID 0, con la confiabilidad de un RAID 1.

Este es el tipo de cosas que se pueden hacer. He aquí los niveles de RAID más comunes:

- RAID 1+0
- RAID 5+0
- RAID 5+1

Debido a que los RAID anidados se utilizan en ambientes más especializados, no nos vamos a ir en más detalles aquí. Sin embargo, hay dos puntos a tener en mente cuando se piense sobre

RAID anidados:

- Otros aspectos - El orden en el que los niveles RAID son anidados pueden tener un gran impacto en la confiabilidad. En otras palabras, RAID 1+0 and RAID 0+1 no son lo mismo.
- Los costos pueden ser altos - Si hay alguna desventaja común a todas las implementaciones RAID, es el costo; por ejemplo, la formación RAID 5+1 más pequeña posible, consiste de seis discos (y se requieren hasta más discos para formaciones más grandes).

Ahora que ya hemos explorado los conceptos detrás de RAID, vamos a ver cómo se puede implementar RAID.

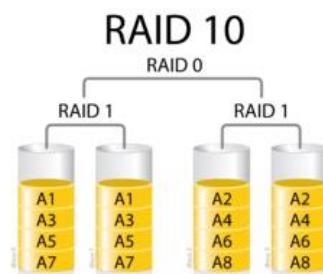


Figura 6 Imagen representativa de un RAID 1+0. Aquí vemos como se combinan distintos tipos de RAID, para generar una formación de discos más compleja.

Implementaciones RAID

Es obvio a partir de las secciones anteriores que RAID requiere "inteligencia" adicional sobre y por encima del procesamiento usual de discos de E/S para unidades individuales. Como mínimo, se deben llevar a cabo las tareas siguientes:

- Dividir las peticiones de E/S entrantes a los discos individuales de la formación
- Para RAID 5, calcular la paridad y escribirla al disco apropiado en la formación
- Supervisar los discos individuales en la formación y tomar las acciones apropiadas si alguno falla
- Controlar la reconstrucción de un disco individual en la formación, cuando ese disco haya sido reemplazado o reparado
- Proporcionar los medios para permitir a los administradores que mantengan la formación (eliminando y añadiendo discos, iniciando y deteniendo reconstrucciones, etc.)

Hay dos métodos principales que se pueden utilizar para lograr estas tareas. Las próximas dos secciones las describen en más detalles.

Hardware RAID

Una implementación de hardware RAID usualmente toma la forma de una tarjeta controladora de disco. La tarjeta ejecuta todas las funciones relacionadas a RAID y controla directamente las unidades individuales en las formaciones conectadas a ella. Con el controlador adecuado,

las formaciones manejadas por una tarjeta de hardware RAID aparecen ante el sistema operativo anfitrión como si se tratases de unidades de disco normales.

La mayoría de las tarjetas controladoras RAID trabajan con SCSI, aunque hay algunos controladores RAID basados en ATA también. En cualquier caso, la interfaz administrativa se implementa usualmente en una de tres formas:

- Programas de utilerías especializados que funcionan como aplicaciones bajo el sistema operativo anfitrión, presentando una interfaz de software a la tarjeta controladora
- Una interfaz en la tarjeta usando un puerto serial que es accedido usando un emulador de terminal
- Una interfaz tipo BIOS que solamente es accesible durante la prueba de encendido del sistema

Algunas controladoras RAID tienen más de un tipo de interfaz administrativa disponible. Por razones obvias, una interfaz de software suministra la mayor flexibilidad, ya que permite funciones administrativas mientras el sistema operativo se está ejecutando. Sin embargo, si está arrancando un sistema operativo desde una controladora RAID, se necesita una interfaz que no requiera un sistema operativo en ejecución.

Puesto que existen muchas tarjetas controladoras RAID en el mercado, es imposible ir en más detalles aquí. Lo mejor a hacer en estos casos es leer la documentación del fabricante para más información.



Figura 7 Controladora RAID. Allí podemos ver en la parte superior, los 4 conectores SATA disponibles para conectar hasta cuatro discos rígidos en simultáneo.

Software RAID

Software RAID es RAID implementado como kernel - o software a nivel de controladores para un sistema operativo particular. Como tal, proporciona más flexibilidad en términos de soporte de hardware - siempre y cuando el sistema operativo soporte ese hardware, se pueden configurar e implementar las formaciones RAID. Esto puede reducir dramáticamente el costo de implementar RAID al eliminar la necesidad de adquirir hardware costoso especializado.

A menudo el exceso de poder de CPU disponible para los cálculos de paridad RAID exceden en gran medida el poder de procesamiento presente en una tarjeta controladora RAID. Por lo

tanto, algunas implementaciones de software RAID en realidad tienen mejores capacidades de rendimiento que las implementaciones de hardware RAID.

Sin embargo, el software RAID tiene ciertas limitaciones que no están presentes en hardware RAID. La más importante a considerar es el soporte para el arranque desde una formación de software RAID. En la mayoría de los casos, solamente se puede utilizar formaciones RAID 1 para arrancar, ya que el BIOS del computador no está al tanto de RAID. Puesto que no se puede distinguir una unidad única de una formación RAID 1 de un dispositivo de arranque no RAID, el BIOS puede iniciar exitosamente el proceso de arranque; luego el sistema operativo puede cambiarse a la operación desde el software RAID una vez que haya obtenido el control sobre el sistema.

Administración de volúmenes lógicos

Otra tecnología de almacenamiento avanzada es administración de volúmenes lógicos o logical volume management (LVM). LVM hace posible tratar a los dispositivos físicos de almacenamiento masivo como bloques de construcción a bajo nivel en los que se construyen diferentes configuraciones de almacenamiento. Las capacidades exactas varían de acuerdo a la implementación específica, pero pueden incluir la agrupación del almacenamiento físico, redimensionamiento de volúmenes lógicos y la migración de datos.

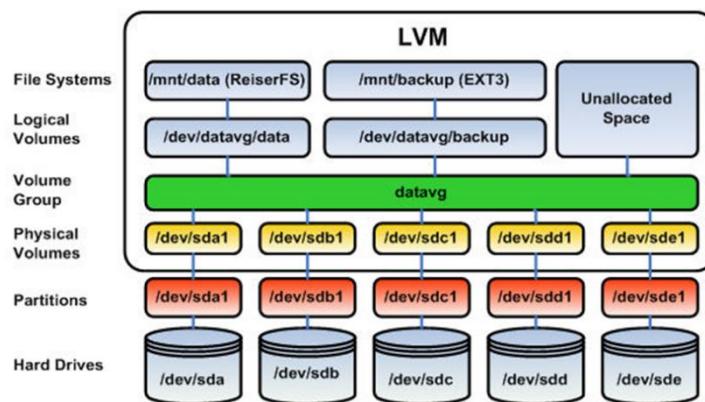


Figura 8 Esquema representativo de la distribución y el agrupamiento con LVM.

Agrupamiento de almacenamiento físico

Aunque los nombres dados a esta capacidad pueden variar, la agrupación del almacenamiento físico es la base para todas las implementaciones de LVM. Como su nombre lo implica, los dispositivos físicos de almacenamiento masivo se pueden agrupar de forma tal para crear uno o más dispositivos lógicos de almacenamiento. Los dispositivos lógicos de almacenamiento masivo (o volúmenes lógicos) pueden ser más grandes en capacidad que cualquiera de los dispositivos físicos de almacenamiento subyacentes.

Por ejemplo, dadas dos unidades de 100GB, se puede crear un volumen lógico de 200GB. Sin embargo, también se pueden crear un volumen lógico de 150GB y otro de 50GB. Es posible

cualquier combinación de volúmenes lógicos igual o menor que la capacidad total (en nuestro ejemplo 200GB). Las posibilidades están limitadas solamente a las necesidades de su organización.

Esto hace posible que un administrador de sistemas trate a todo el almacenamiento como un sólo parque de recursos de almacenamiento, disponible para ser utilizado en cualquier cantidad. Además, posteriormente se pueden añadir unidades a ese parque, haciendo un proceso directo el mantenerse al día con las demandas de almacenamiento de sus usuarios.

Redimensionamiento de volúmenes lógicos

En una configuración de sistemas no LVM, el quedarse sin espacio significa - en el mejor de los casos - mover archivos desde un dispositivo lleno a uno con espacio disponible. A menudo esto significa la reconfiguración de sus dispositivos de almacenamiento masivo. Sin embargo, LVM hace posible incrementar fácilmente el tamaño de un volumen lógico. Asuma por un momento que nuestro parque de almacenamiento de 200GB fue utilizado para crear un volumen lógico de 150GB, dejando el resto de 50GB en reserva. Si el volumen lógico de 150GB se llena, LVM permite incrementar su tamaño (digamos por 10GB) sin ninguna reconfiguración física. Dependiendo del entorno de su sistema operativo, se puede hacer esto dinámicamente o quizás requiera una pequeña cantidad de tiempo fuera de servicio para llevar a cabo el redimensionamiento.

Procesos

***Arquitectura y Sistemas Operativos.
Tecnicatura Superior en Programación.
UTN-FRA***

Autores: Prof. Martín Isusi Seff

Revisores: Prof. Marcos Pablo Russo

Versión: 1



Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

Proceso de arranque en GNU/Linux

Para poder empezar a hablar de procesos en GNU/Linux, y del manejo de los mismos, es necesario conocer cómo es el proceso de arranque del sistema.

El proceso de arranque consta de cuatro etapas:

1. BIOS
2. Bootloader
3. Kernel
4. Init

Primera etapa: BIOS

Esta etapa inicia en el momento en que se enciende la PC. Es allí cuando el BIOS (Basic Input Output System) toma el control del sistema para poder realizar operaciones básicas de hardware (reconocimiento, prueba de la memoria, etc.). Una vez que el BIOS completa las operaciones, se encargará de cargar en memoria el bootloader que inicia la segunda etapa.

Es importante mencionar que esta etapa es independiente del sistema operativo instalado. El BIOS se encuentra en una memoria propia de la placa base de la PC y, si bien este sistema puede ser actualizado desde el sistema operativo, no forma parte del mismo.

Segunda etapa: Bootloader

El bootloader o cargador de arranque, es un programa encargado de iniciar el sistema operativo instalado. Este programa se guarda en una porción del disco llamada MBR (Master Boot Record). El MBR ocupa solo 512 bytes en el disco, de los cuales 2 bytes corresponden al "magic number", 64 bytes a la tabla de particiones y 446 bytes al bootloader. El "magic number" corresponde a un número que sirve de "bandera" y representa el inicio del MBR. La tabla de particiones contiene cada partición del disco y la información de cada una de ellas. Por último, el bootloader será el encargado de iniciar el resto del sistema. El bootloader puede contener la información de distintos sistemas operativos, por ejemplo, si una PC tiene instalado Microsoft Windows y Ubuntu, será el bootloader el encargado de mostrar una lista de los sistemas operativos instalados y luego iniciar el sistema deseado.

En GNU/Linux, existen dos bootloaders principales: LILO y GRUB. Si bien LILO prácticamente ya no se usa, es interesante explicar sus características. LILO sólo soporta hasta 16 sistemas operativos instalados y no tiene la posibilidad de iniciar alguno desde la red. No contiene una consola interactiva que permita modificar los parámetros de inicio de un determinado sistema operativo y, por último, si luego de instalado nuestro sistema, realizamos un cambio que requiera modificaciones del bootloader, será necesario modificar los archivos de configuración de LILO y reescribir el bootloader en el MBR.

Por otro lado, GRUB es uno de los bootloaders más utilizados para sistemas operativos GNU/Linux. Las características más importantes son la posibilidad de manejar una cantidad ilimitada de sistemas operativos y de bootear por red, contiene una interfaz de línea de

comandos interactiva que permite establecer parámetros al iniciar un sistema operativo y, por último, si se realiza un cambio que requiera modificaciones en la configuración de GRUB, solamente es necesario modificar cambiar los archivos de configuración sin reescribir el MBR.

Tercera etapa: Kernel

Antes de comenzar de lleno en cómo se ejecuta esta etapa, es necesario recordar algunos aspectos generales del kernel.

El kernel es el componente fundamental de cualquier sistema operativo, ya que es el encargado de comunicar los programas que solicitan recursos y el hardware. Asignará los tiempos de ejecución para cada programa, gestionará cada una de las tareas que realiza el sistema operativo, el hardware del equipo y el sistema de archivos utilizado.

El kernel puede presentarse desarrollado en distintas arquitecturas, monolítico, modular o híbrido. La arquitectura de kernel monolítico implica que todos los módulos necesarios para controlar el hardware del equipo se encuentran cargados en un único archivo comprimido, mientras que los kernel desarrollados bajo una arquitectura modular se construyen con cada módulo compilado como un objeto que puede ser cargado o descargado según sea necesario.

El proceso de carga del kernel se realiza en dos etapas: etapa de carga y etapa de ejecución.

El kernel se encuentra almacenado de forma comprimida en la memoria secundaria del sistema (generalmente un disco rígido, pero puede ser un pendrive, etc.), por lo que la etapa de carga del kernel se encargará de descomprimir el kernel y copiarlo entero en la memoria principal (RAM). Además de la carga del kernel en la memoria principal, también se cargarán los drivers necesarios mediante un proceso llamado initrd. Este proceso creará un sistema de archivos temporal que sólo es utilizado durante la fase de carga.

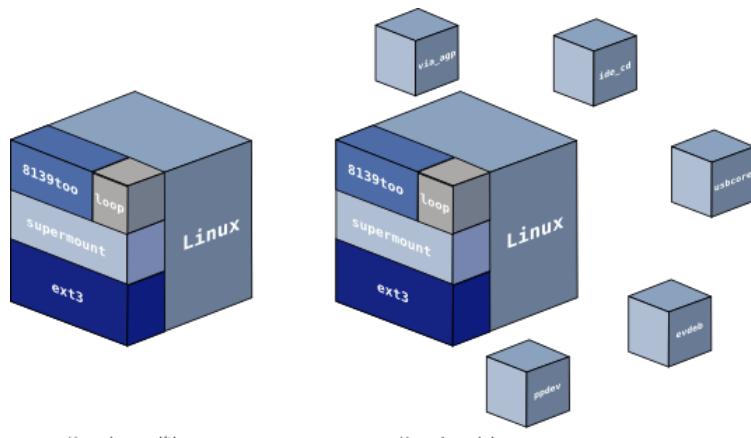


Figura 1

Cuarta etapa: init

Antes de comenzar a hablar del proceso init, debemos definir qué es un proceso. Un proceso es un programa que se ejecuta en un determinado momento en el sistema. Siempre hay una serie de procesos que se ejecutan sin la intervención del usuario y que hacen que el sistema

sea utilizable. Cada proceso contiene un conjunto de estructuras de datos y una dirección en la memoria principal. En esa dirección de memoria se reserva el espacio para que se realice la copia del código del proceso, el área para los datos del mismo, pila del proceso e información adicional utilizada por el sistema durante la ejecución. Existen dos tipos de procesos, los **procesos de usuario**, que son aquellos procesos ejecutados directamente por el usuario, y los **procesos demonio**. Los procesos demonio son aquellos que no requieren la intervención del usuario y se ejecutan en un segundo plano.

Una vez que la etapa de carga y ejecución del kernel se completa, se iniciará el proceso **init**. Este proceso es ejecutado por todos los sistemas basados en Unix y es el responsable de la inicialización de todos los nuevos procesos excepto el proceso swapper. Init se conoce como un proceso *dispatcher* o planificador, encargado de decir qué proceso se ejecutará y cuáles serán copiados/borrados de la memoria principal. A partir del momento de ejecución de init, éste es conocido como el proceso 1. Dentro del directorio /etc se encuentra el archivo inittab que corresponde al archivo de configuración del proceso init.

En GNU/Linux, los procesos se ejecutan de forma jerárquica: cada proceso es lanzado desde un proceso "padre", por lo que el proceso lanzado se llama "hijo". Podemos decir entonces, que todos los procesos ejecutados luego del inicio del sistema son procesos "hijo" de init.

Manejo de procesos

Las estructuras de datos referidas a los procesos contienen información que permite el manejo de éstos. Algunos de los datos que contienen son: mapa de espacio del proceso, estado actual, prioridad de ejecución, máscara actual de la señal del proceso y propietario. Salvo el proceso *init* que tiene el PID 1, todos los demás procesos son creados por otros procesos.

Atributos de un proceso

Algunos de los parámetros asociados a los procesos afectan de forma directa a su ejecución, por ejemplo, el tiempo de proceso, prioridad, ficheros a los que se pueden acceder, etc.

PID (Process IDentification). Es un número que identifica al proceso en el sistema. Se asignan de forma secuencial a cada nuevo proceso que se crea.

PPID (Parent Process IDentification). Es un número que se corresponde con el PID del proceso <padre>. El proceso <padre> es aquel que creó al proceso actual, llamado proceso <hijo> del anterior.

UID, GID (User Identification, Group IDentification). Estos números ya aparecieron en la unidad anterior. Son el número de identificación del usuario que creó el proceso UID, y el número de identificación del grupo de usuario, GID. Sólo el superusuario y el usuario que creó el proceso, llamado propietario del proceso, pueden modificar el estado de operación de los procesos.

EUID, EGID (Effective User IDentification, Effective Group IDentification). Estos números

identifican al usuario que ejecuta el proceso; es a través de estos números y no del UID y GID cómo el sistema determina para qué ficheros el proceso tiene acceso.

Prioridad. La prioridad de un proceso afecta al tiempo y al orden de ejecución de éste por parte del procesador. Un proceso de mayor prioridad que otro significa que en la ejecución de los dos procesos el sistema asignará un período de ejecución mayor para el de mayor prioridad y, además, lo elegirá más veces para ejecutarlo. Si no se indica nada al crear un proceso, éste toma la misma prioridad que la del proceso <padre>. El propietario y el proceso mismo pueden modificar la prioridad, pero siempre en sentido decrecimiento sólo el superusuario no tiene restricciones para cambiar la prioridad de un proceso.

Control de terminal. Son enlaces que determinan de dónde toman la entrada y la salida de datos y el canal de error los procesos durante su ejecución. Si no se usan redirecciones, se utilizan por defecto la entrada y salida estándar.

Creación y ejecución de un proceso

Cuando un proceso quiere crear un nuevo proceso, el primer paso consiste en realizar una copia de sí mismo mediante la llamada del sistema fork. La operación fork crea una copia idéntica del proceso padre, salvo en los siguientes casos:

- El nuevo proceso tiene un PID distinto y único.
- El PPID del nuevo proceso es el PID del proceso padre.
- Se asume que el nuevo proceso no ha usado recursos.
- El nuevo proceso tiene su propia copia de los ficheros descriptores del proceso padre.

Para poder ver los procesos en ejecución en tiempo real, se puede utilizar el comando *top*.

Estado de los procesos

Existen cinco estados posibles que puede adoptar un proceso:

- *Ejecutándose (running R)*. El proceso se está ejecutando.
- *Durmiendo (sleeping S)*. El proceso está en espera de algún recurso del sistema.
- *Intercambiado (swapped)*. El proceso no está en memoria.
- *Zombi (Zombie Z)*. El proceso trata de finalizar su ejecución.
- *Parado (stopped)*. El proceso no puede ser ejecutado.

Cuando un proceso se ejecuta es porque tiene los recursos del sistema necesarios y dispone de tiempo de procesador. El sistema “duerme” un proceso en el momento que necesita recursos del sistema que no se pueden obtener de manera inmediata.

- Los procesos “dormidos” esperan a que ocurra un determinado evento. Por ejemplo, entrada de datos, una conexión de la red, etc. Los procesos “dormidos” no consumen tiempo del procesador.
- Los procesos “intercambiados” no se encuentran en la memoria principal del sistema, se vuelcan a la memoria de intercambio. Esto sucede cuando la memoria principal no dispone de suficiente espacio libre y los datos del proceso pasan continuamente de la memoria de intercambio a la principal como consecuencia, la ejecución del proceso se realiza de forma poco efectiva.
- Cuando un proceso está “zombi” no se puede volver a ejecutar, el espacio de memoria que utilizaba se ha liberado y sólo mantiene algunas de las estructuras de datos que les dan entidad a los procesos.
- Un proceso está <parado> cuando no se puede ejecutar. La diferencia entre un proceso “dormido” y otro “parado” es que este último no puede continuar ejecutándose hasta que reciba una señal CONT de otro proceso. Las siguientes situaciones llevan un proceso al estado de parada:
 - Desde una shell csh se pulsa Ctrl-Z.
 - A petición de un usuario o proceso.
 - Cuando un proceso que se ejecuta en segundo plano trata de acceder a un terminal.

Señales

Las señales se utilizan para que un proceso suspenda la tarea que está realizando y se ocupe de otra. Cuando se envía una señal a un proceso, éste puede actuar de dos maneras: si el proceso dispone de una rutina específica para esa señal, llamada “manejador” o *handler*, la utiliza, en caso contrario, el Kernel utiliza el manejador por defecto para esa señal. Utilizar un manejador específico para una señal en un proceso se denomina capturar la señal.

Hay dos señales que no pueden ser ni capturadas ni ignoradas, *KILL* y *STOP*. La señal de *KILL*

hace que el proceso que la recibe sea destruido. Un proceso que recibe la señal de STOP suspende su ejecución hasta que reciba una señal CONT.

Número	Nombre	Descripción	Por defecto	¿Capturada?	¿Bloqueada?
1	SIGHUP	Hangup.	Terminar	SI	SI
2	SIGINT	Interrumpir.	Terminar	SI	SI
3	SIGQUIT	Salir.	Terminar	SI	SI
4	SIGILL	Instrucción ilegal.	Terminar	SI	SI
5	SIGTRAP	Trazar.	Terminar	SI	SI
6	SIGIOT	IOT.	Terminar	SI	SI
7	SIGBUS	Error de de Bus.	Terminar	SI	SI
8	SIGFPE	Excepción aritmética.	Terminar	SI	SI
9	SIGKILL	Destruir	Terminar	NO	NO
10	SIGUSR1	Primera señal definida por el usuario.	Terminar	SI	SI
11	SIGSEGV	Violación de segmentación.	Terminar	SI	SI
12	SIGUSR2	Segunda señal definida por el usuario.	Terminar	SI	SI
13	SIGPIPE	Escribir en un pipe.	Terminar	SI	SI
14	SIGALRM	Alarma del reloj.	Terminar	SI	SI
15	SIGTERM	Terminación del programa.	Terminar	SI	SI
16	SIGSTKFLT			SI	SI
17	SIGCHLD	El estado del hijo ha cambiado.	Ignorar	SI	SI
18	SIGCONT	Continuar después de parar.	Ignorar	SI	NO
19	SIGSTOP	Parar.	Parar	NO	NO
20	SIGSTP	Parada desde el teclado.	Parar	SI	SI
21	SIGTTIN	Lectura en segundo plano.	Parar	SI	SI
22	SIGTTOU	Escritura en segundo plano.	Parar	SI	SI
23	SIGURG	Condición urgente de socket.	Ignorar	SI	SI
24	SIGXCPU	Excedido el tiempo de CPU.	Terminar	SI	SI

25	SIGXFSZ	Excedido el tamaño de fichero.	Terminar	SI	SI
26	SIGVTALRM	Alarma de tiempo virtual.	Terminar	SI	SI
27	SIGPROF	Alarma.	Terminar	SI	SI
28	SIGWINCH	Cambia de ventana.	Ignorar	SI	SI
29	SIGLOST	Recurso perdido.	Terminar	SI	SI
30	SIGPWR			SI	SI
31	SIGUNUSED			SI	SI

Para poder imprimir esta tabla desde la terminal, se debe correr el comando *kill -l*

Enviar señales a un proceso

Sólo el propietario del proceso y el superusuario (root) pueden mandar señal KILL a un proceso. Por defecto la señal es 15 (TERM). Para enviar una señal a un proceso, se usa el siguiente comando:

```
kill [-señal] PID
```

Usuarios, grupos y permisos

***Arquitectura y Sistemas Operativos.
Tecnicatura Superior en Programación.
UTN-FRA***

Autores: Prof. Martín Isusi Seff

Revisores: Prof. Marcos Pablo Russo

Versión: 1



Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

Los sistemas operativos basados en Linux y Unix trabajan de una manera distinta al resto de los sistemas operativos en términos de “multitarea”, aunque presentan, de todas maneras, algunas diferencias. La principal diferencia radica en que, mientras que la mayoría de sistemas operativos solo permiten que un solo usuario esté trabajando al mismo tiempo, Linux fue diseñado para permitir múltiples usuarios al mismo tiempo. Para que estos usuarios no afecten las sesiones del resto de los usuarios, se tuvo que desarrollar un complejo modelo de permisos.

Permisos de lectura, escritura y ejecución

Los permisos son los “derechos” que tiene un usuario o grupo sobre un archivo o directorio.

Los permisos básicos son *lectura, escritura y ejecución*.

- *Permiso de lectura (read)*. Este permiso hace que el contenido de un archivo sea visible, o, en el caso de los directorios, el que contenido del mismo sea visible.
- *Permiso de escritura (write)*. Este permiso hace que el contenido de un archivo pueda ser modificado, mientras que para directorios permite realizar acciones sobre los archivos que contiene (borrarlos, agregar nuevos archivos, etc.)
- *Permiso de ejecución (execute)*. En archivos, este permiso hará posible la ejecución del mismo. Para esto, el archivo deberá ser un *script* o un programa.

Listar los permisos de un archivo o directorio

Para listar los permisos de un archivo o directorio, basta con ejecutar el comando ***ls***, junto con la opción ***-l***. El resultado de la ejecución de dicho comando mostrará el contenido de un directorio de la siguiente manera:

```
-rw-r--r-- 1 root root 1031 Nov 18 09:22 /etc/passwd
```

Los primeros diez caracteres del ejemplo representan los permisos. El primer carácter (que en este caso es un **-**) representa el tipo de archivo que está listando. Los tipos de archivo son:

- **d** representa directorios
- **s** representa un archivo especial
- **-** representa un archivo regular

Los siguientes nueve caracteres representan puntualmente qué permisos tienen el propietario, los permisos que tienen aquellos usuarios que pertenezcan al mismo grupo que el propietario, y los permisos que tienen aquellos usuarios que no pertenecen al mismo grupo que el propietario. El siguiente cuadro muestra cómo se descomponen los diez caracteres, utilizando el ejemplo anterior:

-	rw-	r--	r--
Tipo de archivo	Permisos del dueño	Permisos del grupo del dueño	Permisos del resto de los usuarios

Podemos observar que los permisos del propietario son ***rw-***. Esto quiere decir que el usuario

tiene permisos de lectura (**r**), y permisos de escritura (**w**). Los permisos de ejecución, si los tuviera, estarían representados con una **x** luego del permiso de escritura. En este caso, como no posee permisos de ejecución, solo vemos un **-**.

Siguiendo con la misma lógica, podemos decir que tanto los usuarios del grupo del propietario, como el resto de los usuarios, solo poseen permisos de lectura.

Administrando cuentas de usuario

El comando para crear un nuevo usuario estándar es **useradd**. La sintaxis es la siguiente:

useradd <nombre>

Las opciones que podemos utilizar son las siguientes:

Opción	Descripción
-d <home_dir>	Nos permite especificar cuál será el directorio principal del usuario que estamos creando.
-e <fecha>	Permite especificar una fecha en la que expirará la cuenta del nuevo usuario.
-s	Especifica el intérprete de línea de comandos que usará por defecto el nuevo usuario. Por ejemplo: useradd nuevousuario -s /bin/bash

Una vez creado el nuevo usuario, es necesario configurar un password para el mismo. Esto se hace utilizando el comando **passwd**. La sintaxis es la siguiente:

passwd <usuario>

Para poder ejecutar el comando, es necesario tener permisos de superusuario.

Otra manera de crear usuarios es utilizar el comando **adduser**. El mismo no se encuentra siempre instalado, por lo que hay que instalarlo como cualquier otro paquete. Este comando permitirá crear usuarios de una manera más simple ya que automáticamente creará el directorio principal, asignará un intérprete de línea de comandos y configurará el password.

Para remover una cuenta de usuario, el comando a utilizar es **userdel**. La sintaxis del mismo es:

userdel <usuario>

Se debe tener en cuenta que ejecutar el comando como se muestra en el ejemplo anterior solo eliminará la cuenta, pero los archivos y el directorio principal no serán borrados. Para poder borrar un usuario completamente, el comando deberá ser ejecutado utilizando la opción **-r**.

userdel -r <usuario>

Entendiendo sudo

El usuario **root**, es el usuario que tiene los permisos para hacer cualquier cosa en el sistema. Es por eso que, por motivos de seguridad, se suele utilizar **sudo**. Este comando permite a

usuarios y grupos, tener acceso a comandos que no podría acceder normalmente. Esto quiere decir que aquellos usuarios ***sudo*** podrán ejecutar comandos como si fuesen *root* sin realmente estar logueados como usuarios *root*.

El paquete ***sudo*** no está presente en todas las distribuciones de GNU/Linux, por lo tanto, en muchos casos será necesario instalarlo.

Para que un usuario determinado sea pueda utilizar ***sudo***, será necesario que el mismo esté presente en el archivo ***sudoers***. Este archivo se encuentra ubicado dentro del directorio ***/etc***. Es importante tener en cuenta que, para agregar un usuario a la lista de ***sudoers*** (agregarlo al archivo), es necesario tener permisos *root*.

Trabajando con grupos

Los grupos en Linux son una manera de organizar los usuarios, principalmente como una medida de seguridad. El control de los grupos se administra a través del archivo ***/etc/group***, que contiene la lista de grupos y sus miembros. Cada usuario tiene un grupo primario por defecto. Cuando un usuario inicia sesión, se asocia al mismo con el grupo primario al que pertenece. Esto quiere decir que cada vez que el usuario cree un archivo, o ejecute un programa, esto quedará asociado al grupo primario. Para cambiar el grupo con el que un usuario está asociado en un momento determinado, se utiliza el comando ***chgrp***.

Cambiando permisos de archivos y directorios

Suponiendo que ejecutamos el comando ***ls -s***, si la salida es la siguiente:

```
-rw-r--r-- 1 root root 1031 Nov 18 09:22 /etc/passwd
```

Analicemos la información. Los primeros diez caracteres sabemos que representan los permisos que distintos grupos, y el propietario, tienen sobre el archivo o directorio. Luego, el número que se muestra representa, en caso de archivos, la cantidad de inodos que utiliza, y en en directorios, representa la cantidad de archivos y subdirectorios que contiene. Luego tendremos el propietario, seguido por el grupo del propietario (al momento que creó el archivo o directorio). En este caso, tanto el propietario, como el grupo, son *root*.

Comando ***chmod***

El comando ***chmod*** nos permitirá cambiar permisos en archivos o directorios. Existen dos maneras de especificar nuevos permisos al utilizar el comando ***chmod***, con letras o números (en base octal).

Para utilizar el comando ***chmod*** con letras, hay que tener en cuenta lo siguiente: El signo ***+*** agregará permisos y el signo ***-*** quitará permisos. Por otro lado, las letras que podemos utilizar y su significado son las siguientes:

- ***r*** representa el permiso de lectura.
- ***w*** representa el permiso de escritura.

- **x** representa el permiso de ejecución.
- **X** representa el permiso de ejecución (sólo aplica a directorios).

Por ejemplo, el comando

chmod +r <archivo>

Nos permitirá agregar permisos de escritura tanto al propietario como a su grupo y al resto de los grupos.

En el caso de utilizar el modo *octal* para definir los permisos, se deberá “calcular” el valor de los permisos tanto para el propietario, como para su grupo y para el resto de los grupos. Este cálculo se basa en la siguiente tabla:

Valor octal	Lectura (r)	Escritura (w)	Ejecución (x)
7	r	w	x
6	r	w	-
5	r	-	x
4	r	-	-
3	-	w	x
2	-	w	-
1	-	-	x
0	-	-	-

Esto se puede traducir en el siguiente ejemplo: Supóngase que se desea agregar los siguientes permisos sobre un archivo llamado **archivo.txt**: permisos de lectura, escritura y ejecución (rwx) al propietario, lectura y escritura (rw-) al grupo del propietario y solo lectura al resto de los usuarios (r--). El comando para realizar esto sería:

chmod 764 archivo.txt

Permisos adicionales en archivos

Además de los permisos comunes, lectura, escritura y ejecución, hay algunos permisos adicionales que también pueden resultar útiles.

Estos permisos adicionales son el **sticky bit (t)** y **setuid bit (s)**. Estos dos permisos describen el comportamiento de los archivos y ejecutables en situaciones “multiusuario”.

Cuando se define en un archivo o directorio, el modo **sticky bit**, significa que solo el propietario (o *root*) pueden eliminar el archivo, independientemente de los permisos que tengan el resto de los usuarios. Para configurar el permiso **sticky bit** sobre un archivo ejecutamos lo siguiente:

chmod +t <archivo>

El permiso **setuid** se puede explicar mediante el siguiente ejemplo: Supóngase el usuario *usuario1* que es propietario de *programa.sh*. El usuario *usuario1*, pertenece al grupo *grupo1*.

Suponiendo que el resto de los usuarios pertenecientes al *grupo1* tienen permisos de ejecución sobre el archivo *archivo1.sh*, éstos podrán ejecutar el archivo como si fueran el *usuario1*. Para aquellos que estén acostumbrados al sistema operativo Windows, este comportamiento es similar a la opción “ejecutar como”, que permite ejecutar un programa como si fuera un usuario distinto.

Cambiando el propietario de un archivo

Por defecto, el propietario de un archivo es aquel que crea el mismo, y por defecto, el grupo asociado es grupo en que esté registrado el usuario al momento de crear el archivo. Para cambiar el propietario de un archivo, se utiliza el comando **chown**. Por ejemplo, se tiene el archivo *lista.txt* con los siguientes detalles:

```
-rw-r--r-- 1 administrador1 grupoadmin 1031 Nov 18 09:22 /home/admin/lista.txt
```

Si se quisiera cambiar el propietario de tal archivo (que actualmente es *administrador1*, y está bajo el grupo *grupoadmin*) por *administrador2*, el comando a ejecutar sería el siguiente:

```
chown administrador2:grupoadmin /home/admin/lista.txt
```

Protocolos

***Arquitectura y Sistemas Operativos.
Tecnicatura Superior en Programación.
UTN-FRA***

Autores: Prof. Martín Isusi Seff

Revisores: Prof. Marcos Pablo Russo

Versión: 1



Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional.

¿Qué es un protocolo?

Los protocolos son mecanismos que definen las reglas y convenciones para la comunicación entre dispositivos. Los protocolos incluyen tanto los mecanismos para identificar y realizar conexiones dentro de una red, así como también especifican las reglas y el formato con el que deben transmitirse los datos. Existen protocolos que, además, incluyen soporte para la compresión de los datos, acuse de recibo de mensajes, etc.

Hoy en día, los protocolos de red, generalmente utilizan un mecanismo llamado conmutación de paquetes para enviar y recibir los mensajes en forma de paquetes. Esto significa, mensajes divididos en piezas más pequeñas, que luego de enviadas, son reensambladas en el dispositivo destino.

Al hablar de protocolos, es importante mencionar el modelo OSI. Este modelo define un estándar para las comunicaciones entre distintos dispositivos. Siete capas son las que forman partes de este modelo:

- Capa de aplicación
- Capa de presentación
- Capa de sesión
- Capa de transporte
- Capa de red
- Capa de datos
- Capa física

El propósito de este apunte es desarrollar teórica y prácticamente algunos protocolos comunes de la capa de aplicación (HTTP, FTP, etc.). Esta capa define todos los protocolos que describen la comunicación entre aplicaciones (cómo transmitir archivos, cómo realizar una consulta a una página web, etc.). Puede decirse que la capa de aplicación es la que se encuentra en un “nivel más alto” con respecto al hardware, ya que no importa de qué manera se transmitan a bajo nivel los paquetes, estos protocolos funcionan de igual manera.

Funciones de los protocolos de la capa de aplicación

Tanto el dispositivo emisor como el receptor utilizan protocolos de capa de aplicación durante una comunicación. Para que la comunicación sea exitosa, todos los dispositivos que participan deben comunicarse con el mismo protocolo, esto quiere decir que, quién envía utilizará un protocolo para formar los paquetes y, el receptor, deberá utilizar el mismo protocolo para recrear el mensaje enviado.

Los protocolos de la capa de aplicación deben realizar las siguientes tareas:

- Establecer reglas consistentes para el intercambio de datos entre las aplicaciones y servicios cargados en el emisor y receptor.
- Especificar cómo se estructuran los datos, y los tipos de mensajes que pueden enviarse los

participantes de la comunicación.

Muchos tipos diferentes de aplicaciones se comunican a través de la red. Es por eso que, la capa de aplicación debe implementar múltiples protocolos para los distintos tipos de comunicaciones. Cada protocolo tiene un propósito específico y contiene las características requeridas por tal propósito.

Las aplicaciones y servicios pueden utilizar múltiples protocolos a lo largo de una sola conversación. Por ejemplo, un protocolo puede especificar cómo establecer la comunicación, mientras que otro describe el proceso para transferir los datos.

Modelo cliente-servidor

En el modelo cliente-servidor, dos dispositivos forman parte de la comunicación. Un dispositivo que solicita información llamado **cliente**, y un dispositivo que provee tal información llamado **servidor**. El cliente inicia la comunicación o intercambio, enviando un pedido (**request**). Este pedido puede contener información requerida por el servidor para generar la respuesta (**response**).

Protocolo HTTP

El protocolo HTTP (HyperText Transfer Protocol) es un protocolo **sin estado**, que forma parte de la capa de aplicación y permite la comunicación entre sistemas distribuidos. Este protocolo es la base de la web como la conocemos ahora.

HTTP permite la comunicación entre clientes y servidores bajo distintas configuraciones de red. Esto es posible ya que no se necesita conocer ningún parámetro particular o configuración de los sistemas que se están comunicando. Se dice que es "sin estado", ya que durante una comunicación en la que se transmiten distintos mensajes, ninguno de estos mensajes comparte un estado en común, esto quiere decir que son independientes entre sí.

Dado que un dispositivo generalmente posee una sola comunicación física con la red, necesita existir una manera de diferenciar las distintas conexiones que se producen de distintas aplicaciones en un mismo equipo. Esta distinción se realiza a través de la asignación de **puertos** a cada una de las aplicaciones. Las aplicaciones que trabajan con el protocolo HTTP utilizan de manera estándar el puerto 80, aunque puede haber excepciones.

La comunicación entre el servidor y el cliente ocurre a través de un par **request/response**.

La *request* o *peticIÓN* es la iniciadora de la comunicación, y la realiza el cliente a través de una dirección **URL** (Uniform Resource Locator). Un ejemplo de URL es:

http://www.dominio.com:8080/pagina/archivo.php?a=1&b=2

Este ejemplo podemos descomponerlo en los siguientes componentes:

Protocolo	Servidor o <i>host</i>	Puerto	Ruta al recurso	Consulta
http://	www.dominio.com	8080	/pagina/archivo.php	?a=1&b=2

Las URL son las identidades del *host* con el que nos queremos comunicar. Cuando utilizamos el protocolo HTTP, con especificar la URL no es suficiente. Además de la URL para “ubicar” el *host*, es necesario especificar el **método** que queremos utilizar. El *método* especifica la acción que debe realizar el *host* en base a la petición del cliente. Los cuatro *métodos* más comunes son:

- **GET.** Este método está pensado para obtener un recurso existente del servidor, aunque no es exclusivamente para ello. Cuando se utiliza el método GET para una petición, toda información extra que necesite ser enviada al servidor se codificará en la URL. En el siguiente ejemplo se hace una petición al recurso *traerUsuario.php* del *host* *usuarios.com*, y se pasan, como información extra, dos parámetros: *nombre* y *apellido*.
http://usuarios.com/traerUsuario.php?nombre=Marcelo&apellido=Rodriguez
- **POST.** Este método está pensado para la creación de un nuevo recurso en el servidor. Al igual que el método GET, el método POST suele utilizarse pasando parámetros al servidor, pero a diferencia de GET, en POST los parámetros no se codifican como parte de la URL, sino que viajan en el **cuerpo** de la petición. Esto hace que sea un método más seguro, ya que la información no es visible directamente para el usuario.
- **PUT.** Este método se utiliza para actualizar un recurso. Suele ir acompañado de información al igual que GET y POST.
- **DELETE.** Se utiliza para eliminar recursos existentes.

Suele decirse que los métodos PUT y DELETE son una versión específica del método POST, ya que este puede utilizarse también para la actualización y eliminación de recursos.

Respuestas y códigos de status

Una vez que el cliente realiza la petición (*request*), el servidor realiza las operaciones correspondientes y devuelve una respuesta (*response*). Las respuestas del servidor llevan consigo un código que representa cómo se completó la operación, y de qué manera la misma debe ser interpretada por el cliente.



Figura 1

Códigos 2xx

Los códigos 2xx indican que la petición fue procesada correctamente por parte del servidor. El código más común cuando una petición se procesa correctamente es el código **200**. En el caso

que el usuario haya realizado una petición por un recurso (utilizando el método GET), el recurso pedido será devuelto en el **cuerpo** de la respuesta. Otros códigos que indican un procesamiento correcto son:

- **202.** La petición de un recurso fue aceptada, pero el mismo puede no estar incluido en la respuesta.
- **204.** Representa una respuesta que no tienen ningún contenido.
- **205.** Indica al cliente que vuelva a mostrar el contenido de cero.
- **206.** Indica que la respuesta contiene solo una parte de la respuesta.

Códigos 3xx

Los códigos 3xx indican al cliente que debe realizar alguna acción extra. El caso más común es que el servidor indique al cliente que debe redireccionarse a una dirección distinta.

- **301.** El recurso fue movido permanentemente (cambia su URL).
- **303.** El recurso fue movido temporalmente y el usuario debe redirigirse. Como parte de la respuesta, se envía la URL temporal.
- **304.** El servidor determina que el recurso requerido por el cliente no se ha modificado desde la última vez que lo consultó, por lo tanto, indica que debe utilizar la copia almacenada en *caché*.

Códigos 4xx

Los códigos 4xx son indican al cliente que hay un error en la petición. Esto puede suceder cuando se solicita un recurso que no existe o hay algún problema en la petición misma.

Algunos de los códigos 4xx más comunes son:

- **400.** Indica que la petición está mal realizada (se conoce como *400 Bad Request*).
- **401.** Indica la necesidad de una autenticación del cliente.
- **403.** Indica que el servidor niega al cliente el acceso al recurso solicitado.
- **404.** Indica que el recurso solicitado no fue encontrado. (*404 not found*)
- **405.** Indica que el método utilizado en la petición (GET, POST, etc.) no está permitido, o el servidor no lo soporta.

Códigos 5xx

Estos códigos son utilizados para indicar un fallo en el servidor durante el procesamiento de la petición.

- **500.** Indica un error interno en el servidor (*Internal server error*)
- **501.** Indica que el servidor todavía no soporta la operación pedida por el cliente (*Not implemented*)
- **503.** Este código se devuelve cuando existe un problema interno en el servidor, o cuando el mismo está sobrecargado. En estos casos, suele suceder que el servidor

directamente no devuelve ninguna respuesta. Cuando el servidor no da respuesta, se consume el tiempo de la petición y se produce un **timeout**.

Protocolo FTP

El protocolo FTP (*File Transfer Protocol*) es uno más dentro de la capa de aplicación, y se utiliza para la transferencia de archivos a través de la red.

Un servidor que ofrezca un servicio FTP, nos permitirá acceder a directorios y archivos en un determinado directorio, o la raíz de directorios completa, dependiendo de los permisos que tengamos. Dicho esto, es necesario mencionar que para conectarnos a un servidor FTP, es necesario contar con un usuario y contraseña. Si bien existe la posibilidad de configurar el servidor de tal manera que acepte conexiones **anónimas**, esto no es recomendado por motivos de seguridad. Si se configura un usuario anónimo, será necesario prestar atención a los permisos que se conceden al mismo.

Así como HTTP tiene un puerto por defecto (puerto 80), el protocolo FTP trabajará por defecto en el **puerto 21**.

Existen múltiples maneras de conectarse a un servidor FTP.

- **Conexión desde un navegador.** Para conectarse a un servidor FTP a través de un navegador, tendremos que utilizar una URL, al igual que en HTTP. La URL debe tener la siguiente forma:

`ftp://usuario@dominioftp.gov/`

A diferencia de una URL para acceder a un recurso a través de HTTP, aquí el protocolo que se especifica en la dirección es `ftp://`. Luego será necesario especificar el nombre de usuario con el que se desea realizar la conexión, seguido del carácter `@` y el dominio del servidor.

- **Conexión desde la línea de comandos.** En la mayoría de los sistemas operativos existe un comando para establecer conexiones FTP.
- **Conexión a través de una interfaz gráfica.** Además de la línea de comandos y el navegador, existen aplicaciones gráficas que nos permiten acceder a un servidor FTP y navegar los archivos. Generalmente estas aplicaciones permiten otras operaciones como modificación de los archivos en el servidor, copia, eliminación, etc. Si bien esto es posible realizarlo íntegramente desde la línea de comandos, una interfaz gráfica nos simplificará las tareas.

Protocolo SSH

El protocolo SSH (*Secure Shell*), es un método para iniciar una sesión segura, en una computadora remota, utilizando distintos métodos como autenticación con passwords, llaves SSH, etc. Una vez iniciada la sesión, la información transmitida desde el servidor hacia el cliente, y viceversa.

El protocolo SSH puede utilizarse para:

- Proveer acceso a usuarios y procesos automáticos a un servidor remoto.
- Transferencia de archivos. Si bien se puede utilizar FTP para ello, SSH ofrece una alternativa más segura.
- Ejecutar comandos de manera remota.

Existen varias muchas maneras de conectarse a un servidor utilizando el protocolo SSH. En GNU/Linux o sistemas operativos basados en Unix, se puede utilizar el comando `ssh` especificando el servidor al que se desea conectar. En Microsoft Windows, existen distintas aplicaciones que permiten establecer una comunicación SSH.

Redes P2P

Las redes P2P, son aquellas que conectan múltiples clientes entre sí (también llamados **pares** o **peers**), de una manera determinada y con un fin específico. Este tipo se utiliza generalmente para compartir archivos; aplicaciones, archivos de video, imágenes, documentos, etc. A diferencia de las redes tradicionales cliente-servidor, donde hay siempre un servidor (o múltiples servidores, pero que sirven para un mismo propósito), y múltiples clientes, en las redes P2P, cada nodo (dispositivo conectado a la red), tiene el rol de cliente y servidor al mismo tiempo. Esto hace que cada cliente pueda estar consumiendo recursos de otro cliente o que esté sirviendo recursos para un cliente remoto.