Tutorial básico de GNU/Linux

Javier Smaldone - `http://www.smaldone.com.ar`

v1.0, junio de 2006

El presente tutorial es una guía de aprendizaje de la operación básica del sistema operativo *GNU/Linux* y los entornos *Unix* en general.

Índice

1. Introducción

1.1. Objetivo

En los últimos años, la gran evolución de las interfaces gráficas para *GNU/Linux* y sus herramientas asociadas, han permitido a los usuarios prescindir de los conocimientos sobre comandos y herramientas tradicionales de *Unix* para su operación. Lamentablemente, esto conlleva el sacrificio de uno de los mayores poderes de este entorno, sobre todo para quienes desean utilizar el sistema para ser algo más que una herramienta de oficina o de diversión.

Es entonces el principal objetivo de este tutorial, introducir de forma breve al usuario novato en el poderoso mundo de las herramientas del entorno *Unix* de la mano del sistema operativo *GNU/Linux*.

El autor recomienda ampliamente seguir el tutorial probando, en la medida de lo posible, cada comando en un sistema real; como así también la visita de los distintos enlaces incluidos en el texto.

1.2. Créditos

Este tutorial está basado, fundamentalmente, en el capítulo 3 de la primera edición del libro *"Linux: Installation and Getting Started"* (Copyright © 1992-1996 Matt Welsh).[1]

El autor agradecerá comentarios, críticas, correcciones o cualquier tipo de aporte a este breve tutorial. Si desea comunicarse con él, puede hacerlo escribiéndole a `javier .ARROBA. smaldone.com.ar`.

1.3. Licencia

Este es un documento libre; puede reproducirlo o modificarlo bajo los términos de la versión 2 (o posteriores, si lo prefiere) de la *GNU General Public License* (Licencia Pública general de la GNU, GNU GPL), tal y como ha sido publicada por la *Free Software Foundation* (FSF).[2]

Este texto se distribuye esperando que sea útil, pero SIN GARANTÍA ALGUNA; e incluso sin la garantía implícita de SER COMERCIALIZABLE o de VALIDEZ PARA UN PROPÓSITO CONCRETO.

1.4. Notación y convenciones

En el presente tutorial usaremos las siguientes convenciones tipográficas:

- Tanto el texto mostrado por el sistema, como los nombres de comandos y las órdenes aparecerán en fuente `typewriter`. Ejemplo: `ls /etc`
- Las órdenes que el usuario ejecutará en el sistema aparecerán en fuente `typewriter` resaltada en *itálica*. Ejemplo: *`ls /etc`*
- Las teclas aparecerán entre corchetes y en fuente `typewriter`. Ejemplo: `[Enter]`
- Las combinaciones de teclas serán denotadas con un signo "+". Ejemplo: `[Ctrl]+[D]`

2. ¿Qué son *Unix* y *GNU/Linux*?

2.1. *Unix*, *Linux* y *GNU*

Unix es uno de los sistemas operativos más populares del mundo debido a su extenso soporte y distribución.[3] Originalmente fue desarrollado como sistema multitarea de tiempo compartido para mini-computadoras y mainframes a mediados de los 70 en los laboratorios de *AT&T*, y desde entonces se ha convertido en uno de los sistemas más utilizados.

¿Cuál es la verdadera razón de la popularidad de *Unix*? Muchos "*hackers*"[4] consideran que es el auténtico y único sistema operativo. El desarrollo de *GNU/Linux* parte de un grupo en expansión de hackers que quisieron hacer un sistema operativo libre con sus propias manos.

Existen numerosas versiones de *Unix* para muchos sistemas, desde computadoras personales hasta supercomputadoras como la Cray Y-MP. La mayoría de las versiones de *Unix* son muy costosas.

[1] Dicho libro, además de abundante material sobre *GNU/Linux* y otros temas relacionados puede ser obtenido desde el sitio del *Linux Documentation Project* (*http://www.tldp.org/*).

[2] El texto completo de la licencia (en inglés) puede verse en *http://www.gnu.org/copyleft/gpl.html* .

[3] Ver *http://es.wikipedia.org/wiki/Unix*

[4] El término "*hacker*" es comúnmente utilizado para referirse a los fanáticos de la programación y las computadoras. Muchas veces se comete el error de llamar hackers a los piratas informáticos. Ver *http://es.wikipedia.org/wiki/Hacker*

Linux es una versión del "*kernel*" (núcleo) de *Unix* de libre distribución[5], inicialmente desarrollado de forma independiente por Linus Torvalds en Finlandia.[6] Luego fue y es desarrollado con la ayuda de muchos programadores y expertos de todo el mundo, comunicados a través de Internet. Cualquiera puede acceder a *Linux* y desarrollar nuevos módulos o cambiarlo a su antojo, ya que es libre (esto es mucho más importante aún que su gratuidad). El kernel *Linux* no utiliza ni una sola línea del código original del *Unix* de *AT&T* o de cualquier otro software privativo, y se distribuye bajo la licencia *GNU GPL*.[7] de la *Free Software Foundation*[8]

En Marzo de 1992 apareció la primera versión "oficial" de *Linux*. Hoy es ya un kernel completo, capaz de ejecutar las herramientas de *GNU* y muchos otros programas. Mucho software libre y no libre ha sido ya portado a *Linux*, y el hardware soportado es mucho mayor que en las primeras versiones (se han desarrollado versiones de *Linux* para más de 15 plataformas, entre las cuales se encuentran Macintosh, SGI, Sparc, Alpha, MIPS, entre otras).

El *Proyecto GNU*[9] fue iniciado en 1984 por Richard M. Stallman (RMS) con el propósito de desarrollar un sistema operativo compatible con *Unix* que fuera software libre. Aunque en la actualidad no ha logrado producir un kernel estable, sus numerosas herramientas se utilizan con el kernel *Linux*. Existen varias distribuciones de *GNU* con *Linux* (usualmente mal llamadas "*distribuciones de Linux*") que ofrecen distintos mecanismos de instalación, colecciones de aplicaciones y herramientas de administración del sistema (Debian, Red Hat, SuSE, etc.).

2.2. Características de *GNU/Linux*

GNU/Linux es un sistema operativo completo con multitarea y multiusuario (como cualquier otra versión de *Unix*). Esto significa que pueden trabajar varios usuarios simultáneamente en él, y que cada uno de ellos puede tener varios programas en ejecución.

Fue desarrollado buscando la portabilidad del código fuente: Encontrará que casi todo el software desarrollado para *Unix* se compila en *GNU/Linux* sin problemas. Y todo lo que se hace para *GNU/Linux* (código del kernel, drivers, librerías y programas de usuario) puede correr sin grandes modificaciones en otros sistemas *Unix*.

GNU/Linux ofrece todo lo necesario para trabajar en red con TCP/IP (el protocolo de Internet). Desde manejadores para las tarjetas de red más populares, PPP (que permite acceder a una red TCP/IP utilizando un módem y la línea telefónica), PPPoE (acceso TCP/IP mediante ADSL), etc. Y también existen gran cantidad de aplicaciones relacionadas con Internet, como navegadores, clientes de correo, clientes de mensajería instantánea, etc.

2.3. La filosofía de *Unix*

En un sistema *Unix* casi todo es un archivo: dispositivos (placa de sonido, discos, impresoras, mouse, teclado, monitor) es un archivo. De esta manera, los programas pueden diseñarse abstrayéndose de las particularidades de los distintos dispositivos, asumiendo que leen y escriben datos en archivos.

[5] Para mayor información sobre el software libre, visite *http://www.gnu.org/philosophy/free-sw.es.html*

[6] Ver

[7] Básicamente, esta licencia establece que el software en cuestión debe ser distribuído incluyendo todo el código fuente y la documentación. Establece además que cualquier persona puede modificar el software de acuerdo a sus necesidades e inclusive puede redistribuirlo, siempre y cuando lo haga bajo la misma licencia. Para mayor información, vea *http://www.gnu.org/licenses/licenses.es.html#TOCGPL*

[8] *http://www.fsf.org/*

[9] *http://www.gnu.org/*

Unix incluye una gran cantidad de pequeñas herramientas (programas) capaces de hacer tareas simples y provee mecanismos para combinarlas (sumando además cualquier programa desarrollado por terceros, hasta por usted mismo), logrando realizar tareas realmente complejas sin mayor esfuerzo (sin requerir, por ejemplo, un programa especializado para tal fin).

Otra característica distintiva de *Unix* es su gran coherencia: el comportamiento y las opciones de los distintos programas son similares. Esta homogeneidad permite transferir fácilmente el conocimiento adquirido respecto de una herramienta a las otras, acelerando notablemente la velocidad de aprendizaje y la productividad.

Una advertencia: Un sistema *Unix* "asumirá" que el usuario sabe lo que hace, y que quiere hacer exactamente lo que ordena. Por lo tanto, no pedirá confirmación a la hora de borrar archivos o realizar alguna otra tarea destructiva o peligrosa. ¡Tenga mucho cuidado a la hora de escribir órdenes peligrosas! En general, los programas de *Unix* son bastante "silenciosos", esto significa que si un programa se ejecuta con éxito (por ejemplo, un comando en donde se especifique el borrado de 200 archivos), finalizará sin mostrar ningún mensaje al usuario.

3. Conceptos básicos de *GNU/Linux*

Bajo *GNU/Linux*, para que los usuarios puedan identificarse en el sistema, deben presentarse ("*log in*") mediante un proceso que consta de dos pasos: Introducir el nombre de usuario ("*login*"), y una contraseña ("*password*"), la cual es su llave personal secreta para entrar en la cuenta. En nuestros ejemplos supondremos que el nombre de usuario es `diego`.

En los sistemas *Unix* tradicionales, el administrador del sistema asignará el nombre de usuario y una contraseña inicial en el momento de crear la cuenta de usuario. Además, cada sistema tiene un nombre ("*hostname*") asignado, que le da nombre a la máquina. El nombre del sistema es usado para identificar computadoras en una red, pero incluso aunque la máquina no esté en red, debería tener su nombre. En nuestros ejemplos, el nombre del sistema será `micasa`.

3.1. Ingresando al sistema

En el momento de presentarse en el sistema, veremos el siguiente indicador en la pantalla:

```
micasa login:
```

Ahora, introducimos nuestro nombre de usuario y presionamos `[Enter]`.[10] En nuestro ejemplo, deberíamos teclear lo siguiente:

```
micasa login: diego
Password:
```

Ahora introducimos la contraseña. Esta no será mostrada en la pantalla conforme se va tecleando, por lo que debe hacer cuidadosamente. Si introducimos una contraseña incorrecta, se mostrará el siguiente mensaje:

```
Login incorrect
```

y deberemos intentarlo nuevamente.

Una vez que hemos introducido correctamente el nombre de usuario y la contraseña, estamos "presentados" en el sistema y listo para iniciar una sesión interactiva y comenzar a trabajar, según los derechos de acceso que nos brinde nuestra cuenta, como veremos más adelante.

[10] En algunos teclados ésta tecla puede aparecer como `[Intro]` o `[Return]`.

3.2. Consolas virtuales

La consola del sistema (o terminal) está formada por el monitor y teclado conectado directamente a la computadora. *GNU/Linux*, proporciona acceso a consolas virtuales (o VCs, por "*Virtual Console*"), las cuales nos permitirán tener mas de una sesión de trabajo activa a la vez desde una única consola física.

Para demostrar esto, ingresamos al sistema (como hemos visto antes). Luego presionamos `[ALT]+[F2]`. Deberíamos ver la línea `micasa login:` de nuevo. Estamos viendo la segunda consola virtual ya que hemos ingresado al sistema por la primera. Para volver a la primera VC, presionamos `[ALT]+[F1]`.

Un sistema *GNU/Linux* recién instalado probablemente nos permita acceder a las primeras seis VCs, usando `[ALT]+[F1]` a `[ALT]+[F6]`, pero es posible habilitar hasta 12 VCs, una por cada tecla de función del teclado.

Mientras que el uso de VC's es algo limitado (después de todo, sólo podemos mirar una por vez), esto debería dar una idea de las capacidades multiusuario del sistema. Mientras estamos trabajando en la VC Nº 1, podemos conmutar a la VC Nº 2 y comenzar a trabajar en otra tarea (inclusive, claro está, con un nombre de usuario diferente), mientras el sistema continúa ejecutando la tarea de la VC Nº 1.

3.3. Comandos e intérpretes de comandos

Un intérprete de comandos (también conocido como "shell") es un programa que toma la entrada del usuario (por ejemplo, las órdenes que teclea) y las traduce a instrucciones del sistema operativo. Esto puede ser comparado con el `COMMAND.COM` de *DOS*, el cual efectúa esencialmente la misma tarea. El intérprete de comandos es sólo una de las interfaces con *Unix*. Hay muchas interfaces posibles, como la interfaz gráfica *X Window*, la cual permite ejecutar comandos usando el ratón y el teclado.

Tan pronto como un usuario ingresa al sistema, se ejecuta un intérprete de comandos y éste ya puede teclear órdenes al sistema. Veamos un ejemplo. Aquí, `diego` entra en el sistema y es situado en el intérprete de comandos:

```
micasa login: diego
Password:
Welcome to micasa!
/home/diego$
```

"`/home/diego$`" es el *prompt* (o indicador) del intérprete de comandos, indicando que está listo para recibir órdenes. Tratemos de decirle al sistema que haga algo interesante:

```
/home/diego$ make love
make: *** No rule to make target ‘love’. Stop.
/home/diego$
```

Bien, como resulta que `make` es el nombre de un programa ya existente en el sistema, el intérprete de comandos lo ejecuta (desafortunadamente, el sistema no está siendo muy amigable).

Esto nos lleva a una cuestión importante: ¿Qué es una orden? ¿Qué ocurre cuando tecleamos "`make love`"? La primera palabra de la orden, "`make`", es el nombre del comando a ejecutar. El resto de la orden es tomado como argumentos (o parámetros) de la comando. Por ejemplo:

/home/diego$ cp hola mundo

Aquí, el nombre del comando es "`cp`", y los argumentos son "`hola`" y "`mundo`".

Cuando se teclea una orden, el intérprete de comandos hace varias cosas. Primero, busca el nombre del comando y comprueba si es un comando interno (es decir, una comando que el propio intérprete de comandos sabe ejecutar por si mismo). Hay bastantes comandos de ese tipo que veremos mas adelante. El intérprete de comandos también comprueba si el comando es un "alias" o nombre sustituto de otro comando. Si no se cumple ninguno de estos casos, el intérprete de comandos busca el programa y lo ejecuta pasándole los argumentos especificados en la línea de comandos.

En nuestro ejemplo, el intérprete de comandos busca el programa llamado `make` y lo ejecuta con el argumento `love`. `make` es un programa usado a menudo para compilar programas grandes, y toma como argumentos el nombre de un "objetivo" a compilar. En el caso de "`make love`", ordenamos a `make` que compile el objetivo `love`. Como `make` no puede encontrar un objetivo de ese nombre, falla enviando un mensaje de error y volviendo al intérprete de comandos.

¿Qué ocurre si tecleamos un comando y el intérprete de comandos no puede encontrar el programa de ese nombre? Bien, probémoslo:

```
/home/diego$ hacer nada
hacer: command not found
/home/diego$
```

Bastante simple, si no se puede encontrar el programa con el nombre dado en la orden (aquí "`hacer`"), se muestra un mensaje de error que debería de ser auto-explicativo. A menudo verá este mensaje de error si se equivoca al teclear un comando (por ejemplo, si hubiese tecleado "`mkae love`" en lugar de "`make love`").

3.4. Saliendo del sistema

Antes de proseguir, deberíamos ver cómo salir del sistema. Desde la línea de comandos usaremos el comando para salir. Hay otras formas, pero esta es la más simple:

```
/home/diego$ exit
```

3.5. Cambiando la contraseña

La primera vez que un usuario ingresa al sistema lo hará utilizando la contraseña asignada por el administrador, pero es altamente recomendable que la cambie de inmediato (además, se recomienda realizar este procedimiento de vez en cuando). El comando `passwd` nos pedirá la contraseña actual y luego la nueva (dos veces, para validarla). Debemos tener cuidado de no olvidar la contraseña, ya que si esto ocurre, el administrador del sistema deberá modificarla por nosotros.

3.6. Archivos y directorios

En la mayoría de los sistemas operativos (*Unix* incluido), existe el concepto de archivo, el cual es un conjunto de información al que se le ha asignado un nombre.

Ejemplos de archivo son un mensaje de correo, o un programa que puede ser ejecutado. Esencialmente, cualquier cosa almacenada en el disco es guardada en un archivo individual.

Los archivos son identificados por sus nombres. Por ejemplo, el archivo que contiene sus números telefónicos podría ser grabado con el nombre "`telefonos`".[11] Estos nombres usualmente identifican el archivo y su contenido de alguna forma significativa para usted. No hay un formato estándar para los nombres de los archivos como lo hay en *DOS* y en otros sistemas operativos; en general estos pueden contener cualquier carácter (excepto "`/`"), y están limitados a 256 caracteres de longitud.

[11] Generalmente trataremos de utilizar letras minúsculas y sin acentos en los nombres de archivos y directorios.

Con el concepto de archivo aparece el concepto de directorio. Un directorio es contenedor. Puede ser considerado como una “carpeta” que contiene muchos archivos diferentes. Tienen nombre con el que los podemos identificar y forman una estructura de árbol; es decir, pueden contener a otros directorios.

Un archivo puede ser referenciado por su nombre con camino, el cual esta constituido por su nombre, antecedido por el nombre del directorio que lo contiene. Por ejemplo, supongamos que `diego` tiene un directorio de nombre `articulos` que contiene tres archivos: `historia`, `ingles` y `tesis` (cada uno de los tres archivos contiene información sobre tres de los proyectos en los que está trabajando). Para referirse al archivo `ingles`, puede especificar su camino: `articulos/ingles`

Como podemos ver, el directorio y el nombre del archivo van separados por un carácter “/”. Por esta razón, los nombres de archivo no pueden contener este carácter. Los usuarios de *DOS* encontrarán esta convención familiar, aunque en ese sistema operativo se usa el carácter “\”.

Como hemos mencionado, los directorios pueden anidarse uno dentro de otro. Por ejemplo, supongamos que `diego` tiene otro directorio dentro de `articulos` llamado `notas`, y dentro de ese directorio, tiene un archivo llamado `enlaces`. El camino de este archivo sería: `articulos/notas/enlaces`

Por lo tanto, el camino realmente es la “ruta” (*path*) que se debe recorrer para localizar a un archivo. El directorio sobre un subdirectorio dado es conocido como el directorio padre. Aquí, el directorio `articulos` es el padre del directorio `notas`.

3.7. El árbol de directorios

La mayoría de los sistemas *Unix* tienen una distribución de archivos estándar, de forma que los recursos y archivos puedan ser fácilmente localizados. Esta distribución forma el árbol de directorios, el cual comienza en el directorio “/”, también conocido como “raíz” o “*root*”.[12] Directamente por debajo (dentro) de / hay algunos subdirectorios importantes: `/bin`, `/etc`, `/dev` y `/usr`, entre otros. Estos a su vez contienen otros directorios con archivos de configuración del sistema, programas, etc.

En particular, cada usuario tiene un directorio “*home*”. Este es el directorio en el que el usuario guardará sus archivos. En los ejemplos anteriores, todos los archivos de `diego` (como `enlaces` y `historia`) estaban contenidos en su directorio “*home*”. Usualmente, los directorios “*home*” de los usuarios cuelgan de `/home` y son denominados con el nombre del usuario al que pertenecen. Por lo tanto, el directorio “*home*” de `diego` es `/home/diego`.

La siguiente figura muestra un árbol de directorio de ejemplo.

```
/ ___ bin
|____ dev
|____ etc
|____ home ___ carlos
|        |____ diego ___ Mail
|                  |____ articulos ___ notas
|                  |____ cartas
|____ lib
|____ proc
|____ tmp
|____ usr ___ X11R6
        |____ bin
        |____ lib
```

[12] No debemos confundir el directorio “*root*” o “*raíz*” con el usuario “`root`” que es el administrador del sistema, ni con el directorio “*home*” de éste último, ubicado en “`/root`”.

```
|____ local ___ bin
|           |____ etc
|____ man
|____ src ___ linux
|____ tmp
```

3.8. Directorio de trabajo actual

Las órdenes que teclee al intérprete de comandos son dadas en términos del directorio actual de trabajo, en donde estamos situados. Cuando un usuario entra al sistema, su directorio de trabajo se inicial es su directorio "*home*" (`/home/diego` en nuestro caso). Cuando referencie a un archivo puede hacerlo con relación a su directorio de trabajo actual, en lugar de especificar el camino completo del archivo.

Veamos un ejemplo: `diego` tiene el directorio `articulos`, y `articulos` contiene el archivo `historia`.

Si deseamos ver el contenido de ese archivo, podemos usar la orden:

```
/home/diego$ cat /home/diego/articulos/historia
```

El comando `cat` muestra el contenido del archivo. Pero como el directorio de trabajo actual es `/home/diego`, podríamos habernos referido al archivo de forma relativa a su directorio de trabajo actual. La orden sería:

```
/home/diego$ cat articulos/historia
```

Por lo tanto, si un camino comienza (como `articulos/historia`) con un carácter distinto a "`/`", el sistema supone que se está refiriendo al archivo con su posición relativa a su directorio de trabajo. Esto es conocido como "camino relativo".

Por otra parte, si un camino comienza con el carácter "`/`", el sistema interpreta esto como el camino completo al archivo partiendo desde el directorio raíz (`/`). Esto es conocido como "camino absoluto".

3.9. Refiriéndose al directorio "*home*"

Bajo `bash` y `tcsh`, dos de los intérpretes de comandos más utilizados, el directorio "*home*" puede ser referenciado usando el carácter de la tilde ("`~`"). Por ejemplo, la orden:

```
/home/diego$ cat ~/articulos/historia
```

es equivalente a:

```
/home/diego$ cat /home/diego/articulos/historia
```

El carácter "~" es sustituido por el intérprete de comandos con el nombre del directorio "*home*". El uso de la tilde es simplemente un atajo; no existe ningún directorio llamado "~", sólo es una ayuda sintáctica proporcionada por el intérprete de comandos.

4. Primeros pasos en *GNU/Linux*

Antes de comenzar es importante destacar que todos los nombres de archivos y comandos son "*case-sensitive*" (hacen diferencia entre mayúsculas y minúsculas, lo cual no ocurre en sistemas operativos como *DOS*). Por ejemplo, el comando "`make`" es diferente a "`Make`" o "`MAKE`". Lo mismo ocurre en el caso de nombres de archivos o directorios.

4.1. Moviéndonos por el entorno

Ahora que ya podemos presentarnos como usuarios y sabemos como indicar archivos con su camino completo, ¿cómo podemos cambiar nuestro directorio de trabajo?

El comando para movernos por la estructura de directorios es `cd`, abreviación de "cambio de directorio". Hay que destacar, que la mayoría de los comandos *Unix* más usados son de dos o tres letras. La forma de uso del comando `cd` es:

```
cd <directorio>
```

donde `<directorio>` es el nombre del directorio al que queremos ingresar.

Como ya vimos, al entrar al sistema comenzamos en el directorio "*home*". Si queremos ir al subdirectorio `articulos`, deberíamos usar la orden:

```
/home/diego$ cd articulos
/home/diego/articulos$
```

Como podemos ver, el prompt cambia para mostrar el directorio actual de trabajo. Ahora que ya estamos en el directorio `articulos` podemos ver el contenido del archivo `historia` con el comando:

```
/home/diego/articulos$ cat historia
```

Ahora estamos en el subdirectorio `articulos`. Para volver al directorio padre de éste, usamos la orden:

```
/home/diego/articulos$ cd ..
/home/diego$
```

(Nótese el espacio entre "`cd`" y "`..`"). Cada directorio tiene una entrada de nombre "`..`" la cual se refiere al directorio padre. De igual forma, existe en cada directorio la entrada "`.`" la cual se refiere a sí mismo. Así que el siguiente comando nos deja donde estamos:

```
/home/diego/articulos$ cd .
/home/diego/articulos$
```

También pueden usarse nombres con el camino absoluto en el comando `cd`. Para ir al directorio de `carlos` con `cd`, introduciremos la siguiente orden:[13]

```
/home/diego/articulos$ cd /home/carlos
/home/carlos$
```

También, usando `cd` sin argumentos nos llevará a nuestro directorio "*home*":

```
/home/carlos$ cd
/home/diego$
```

[13]Siempre y cuando tengamos permiso para ingresar a dicho directorio, como veremos más adelante.

4.2. Viendo el contenido de los directorios

El simple movimiento por el árbol de directorios es poco útil, necesitamos un nuevo comando: `ls`. Este comando muestra en pantalla la lista de archivos y directorios, si no se indica otra cosa, del directorio activo. Por ejemplo:[14]

```
/home/diego$ ls
Mail
articulos
cartas
/home/diego$
```

Aquí podemos ver que `diego` tiene tres entradas en su directorio actual: `Mail`, `articulos` y `cartas`. Esto no nos dice demasiado, ¿son archivos o directorios? Podemos usar la opción "`-F`" del comando `ls` para obtener mas información.

```
/home/diego$ ls -F
Mail/
articulos/
cartas/
/home/diego$
```

Por el carácter "`/`" añadido al final de cada nombre sabemos que las tres entradas son directorios. La orden "`ls -F`" puede también añadir al final un asterisco ("`*`"), esto indica que es un archivo ejecutable (programa). Si "`ls -F`" no añade nada, entonces es un archivo normal.

Por lo general cada comando *Unix* puede tomar una serie de opciones definidas en forma de argumentos. Estos usualmente comienzan con el carácter "`-`"[15], tal como vimos antes con *"ls -F"*. La opción "`-F`" le dice a `ls` que dé información sobre el tipo de cada entrada.

Si a `ls` le pasamos un nombre de directorio, mostrará su contenido:

```
/home/diego$ ls -F articulos
ingles
historia
tesis
notas/
/home/diego$
```

Para ver un listado más interesante, veamos el contenido del directorio `/etc`.

```
/home/diego$ ls /etc

Images      ftpusers    lpc      rc.new   shells
adm         getty       magic    rc0.d    startcons
bcheckrc    gettydefs   motd     rc1.d    swapoff
brc         group       mount    rc2.d    swapon
brc~        inet        mtab     rc3.d    syslog.conf
csh.cshrc   init        mtools   rc4.d    syslog.pid
```

[14] Nótese que los listados aparecen ordenados alfabéticamente. Las letras mayúsculas son "menores" a las minúsculas.

[15] Similar al uso del carácter "/" en *DOS*.

```
csh.login      init.d         pac            rc5.d          syslogd.reload
default        initrunlvl     passwd         rmt            termcap
disktab        inittab        printcap       rpc            umount
fdprm          inittab.old    profile        rpcinfo        update
fstab          issue          psdatabase     securetty      utmp
ftpaccess      lilo           rc             services       wtmp

/home/diego$
```

Los usuarios de *DOS* notarán que los nombres de los archivos pueden ser más largos que 11 caracteres (8 del nombre y 3 de la extensión) y pueden contener puntos en cualquier posición.[16] Incluso es posible que un archivo contenga más de un punto en su nombre.

Vayamos al directorio raíz con "`cd ..`" y desde allí vayamos al directorio `/usr/bin`:

```
/home/diego$ cd ..
/home$ cd ..
/$ cd usr
/usr$ cd bin
/usr/bin$
```

También podemos movernos dentro de directorios en múltiples pasos, como en "`cd /usr/bin`". Tratemos de movernos por varios directorios usando `cd` y de ver su contenido con `ls`. En algunos casos podremos encontrarnos con el desagradable mensaje de error "Permission denied" (permiso denegado). Esto se debe a cuestiones de seguridad del sistema. Para moverse o listar un directorio debemos de tener una serie de permisos. Hablaremos sobre ello más adelante.

4.3. Creando directorios

Es el momento de aprender a crear directorios. Para ello se usa el comando `mkdir`. Probemos lo siguiente:

```
/home/diego$ mkdir nuevo
/home/diego$ ls -F
Mail/
articulos/
cartas/
nuevo/
/home/diego$ cd nuevo
/home/diego/nuevo$ ls
/home/diego/nuevo$
```

Acabamos de crear un directorio nuevo y movernos dentro de él. Como no contiene ningún archivo, veamos cómo copiar archivos desde un lugar a otro.

4.4. Copiando archivos

La copia de archivos es efectuada por el comando `cp`:

[16] En *Unix* no existe el concepto de "extensión" de un archivo. Es solamente una convención que el nombre de algunos archivos termine con un punto y alguna secuencia de letras (no necesariamente tres) que denoten su tipo.

```
/home/diego/nuevo$ cp /etc/termcap .
/home/diego/nuevo$ cp /etc/shells .
/home/diego/nuevo$ ls -F
shells termcap
/home/diego/nuevo$ cp shells bells
/home/diego/nuevo$ ls -F
bells shells termcap
/home/diego/nuevo$
```

El comando `cp` copia los archivos listados en la línea de comandos al archivo o directorio pasado como último argumento. Nótese que usamos el directorio "`.`" para referirnos al directorio actual.

4.5. Moviendo archivos

El comando `mv` mueve archivos en lugar de copiarlos. La sintaxis es muy sencilla:

```
/home/diego/nuevo$ mv termcap sells
/home/diego/nuevo$ ls -F
bells sells shells
/home/diego/nuevo$
```

Nótese como `termcap` ya no existe, y en su lugar está el archivo `sells`. Este comando puede usarse para renombrar archivos, como acabamos de hacer, pero también para mover archivos a directorios diferentes.

Nota: `mv` y `cp` sobreescribirán los archivos destino (si ya existen) sin consultar. Sea cuidadoso cuando mueva un archivo a otro directorio: puede haber ya un archivo con el mismo nombre que será sobreescrito y su contenido se perderá para siempre.

4.6. Borrando archivos y directorios

Para borrar un archivo, usamos el comando `rm`.

```
/home/diego/nuevo$ rm bells sells
/home/diego/nuevo$ ls -F
shells
/home/diego/nuevo$
```

En el directorio `nuevo` sólo ha quedado el archivo `shells`. Nótese que `rm` por defecto no preguntará antes de borrar un archivo, por lo tanto debemos ser muy cuidadosos.

Un comando relacionado con `rm` es `rmdir`. Éste borra un directorio, pero sólo si está vacío. Si el directorio contiene archivos o subdirectorios, nos informará del error.

4.7. Viendo el contenido de archivos

Los comandos `cat` y `less` son usados para ver el contenido de archivos. `less` muestra el archivo pantalla a pantalla, permitiéndonos movernos hacia adelante y atrás, mientras que `cat` lo muestra completo de una vez.

Para ver el contenido del archivo `shells` podemos usar la orden:

```
/home/diego/nuevo$ less shells
```

Durante la ejecución de `less` podemos usar `[RePág]` y `[AvPág]`[17] para retroceder y avanzar por páginas, y las teclas de cursor hacia arriba y abajo para retroceder y avanzar por líneas. `[Q]` finalizará la ejecución de `less`. Hay otros comandos disponibles, los citados son sólo los más básicos.

Salgamos de `less` y probemos "`cat /etc/termcap`". El texto probablemente pasará demasiado rápido como para poder leerlo. El comando `cat`; viene de "concatenar", que es para lo que realmente sirve, pero también puede ser usado para concatenar el contenido de varios archivos. Esto se verá mas adelante.

4.8. Obteniendo ayuda en línea

Prácticamente cada sistema *Unix* proporciona una utilidad conocida como "páginas de manual". Estas páginas contienen documentación en línea para todos los comandos del sistema, recursos, archivos de configuración, etc. El comando usado para acceder a las páginas de manual es `man`. Por ejemplo, si estamos interesados en conocer otras opciones del comando `ls`, podemos escribir:

```
/home/diego$ man ls
```

y veremos la página de manual para `ls`.

Desafortunadamente para los principiantes, la mayoría de las páginas de manual contienen detalles técnicos del comando sin ningún ejemplo ni explicación adicional acerca de su uso. Pese a esto, estas páginas son una gran fuente de información que permiten refrescar la memoria si olvidamos la sintaxis de un comando.[18] Pruebe `man` con los comandos que ya hemos tratado y con los que vayamos introduciendo. Notará que alguno de los comandos no tiene página de manual. Esto puede deberse a diferentes motivos. El comando puede ser interna del intérprete de comandos (como el caso de `cd`), o un "alias" (renombre de otro comando), en cuyo caso no tendrán una página propia.

5. Sumario de comandos básicos

Esta sección introduce algunas de los comandos básicas más útiles de un sistema *Unix*, incluidos los ya cubiertos en las secciones anteriores.

Nótese que las opciones usualmente comienzan con "`-`" y en la mayoría de los casos se pueden añadir múltiples opciones de una letra con un único "`-`". Por ejemplo, en lugar de usar "`ls -l -F`" es posible usar "`ls -lF`". En lugar de listar todas las opciones disponibles para cada uno de los comandos sólo hablaremos de aquellas más útiles o importantes. De hecho, la mayoría de los comandos tienen un gran número de opciones. Puede usar `man` para ver las páginas de manual de cada comando, la cual le mostrará la lista completa de opciones disponibles.

Nótese también, que la mayoría de los comandos toman una lista de archivos o directorios como argumentos, denotados como "<`archivo1`> `...` <`archivoN`>". Por ejemplo, el comando `cp` toma como argumentos la lista de archivos a copiar, seguidos del archivo o directorio destino. Cuando se copia o mueve más de un archivo, el destino <`archivoN`> debe ser un directorio.

[17] `[PgUp]` y `[PgDown]` en los teclados en inglés

[18] Una respuesta muy común a una pregunta de un principiante en un foro de *Unix* será "RTFM", por "*Read The F*****g Manual*".

5.1. Operaciones sobre directorios

cd

Cambia el directorio de trabajo actual.

Sintaxis: cd <directorio>

<directorio> es el directorio al que cambiamos. (“.” se refiere al directorio actual, “..” al directorio padre.)

Ejemplo: “cd ../nuevo” pone “../nuevo” como directorio actual.

ls

Muestra información sobre los archivos o directorios indicados.

Sintaxis: ls <archivo1> <archivo2> ... <archivoN>

Donde <archivo1> a <archivoN> son los archivos o directorios a listar.

Opciones: Éste comando tiene gran cantidad de opciones. Las más usadas son: -F (muestra información sobre el tipo de archivo) y -l (da un listado “largo” incluyendo tamaño, propietario, permisos, etc.).

Ejemplo: “ls -lF /home/diego” mostrará el contenido del directorio “/home/diego”.

mkdir

Crea directorios.

Sintaxis: mkdir <dir1> <dir2> ...<dirN>

Donde <dir1> a <dirN> son los directorios a crear.

Ejemplo: “mkdir /home/diego/prueba” crea el directorio “prueba” dentro de “/home/diego”.

rmdir

Borra directorios vacíos. El directorio de trabajo actual no debe de estar dentro del directorio a borrar.

Sintaxis: rmdir <dir1> <dir2> ... <dirN>

Donde <dir1> a <dirN> son los directorios a borrar.

Ejemplo: “rmdir /home/diego/articulos” borra el directorio “/home/diego/articulos” si está vacío.

5.2. Operaciones sobre archivos

cp

Copia archivos.

Sintaxis: cp <archivo1> <archivo2> ... <archivoN> <destino>

Donde <archivo1> a <archivoN> son los archivos a copiar y <destino> es el archivo o directorio destino.

Ejemplo: “cp ../primero segundo” copia el archivo “../primero” al archivo o directorio “segundo”.

mv

Mueve archivos. Es equivalente a una copia seguida del borrado del original.Puede ser usado para renombrar archivos.

Sintaxis: mv <archivo1> <archivo2> ... <archivoN> <destino>

Donde <archivo1> a <archivoN> son los archivos a mover y <destino> es el archivo o directorio destino.

Ejemplo: “mv ../primero segundo” mueve el archivo “../primero” al archivo o directorio “segundo”.

rm

Borra archivos (de forma irrecuperable).

Sintaxis: rm <archivo1> <archivo2> ... <archivoN>

Donde <archivo1> a <archivoN> son los nombres de los archivos a borrar.

Opciones: -i pedirá confirmación antes de borrar un archivo.

Ejemplo: “rm -i /home/diego/primero /home/diego/segundo” borra los archivos “primero” y “segundo” en “/home/diego”.

cat

Concatena archivos. También es usado para mostrar el contenido completo de un archivo.

Sintaxis: cat <archivo1> <archivo2> ... <archivoN>

Donde <archivo1> a <archivoN> son los archivos a mostrar.

Ejemplo: “cat cartas/mi-novia” muestra por la pantalla el contenido del archivo mi-novia que se encuentra en el directorio cartas.

5.3. Otras utilidades

less

Muestra el contenido de los archivos indicados, una pantalla cada vez.

Sintaxis: less <archivo1> <archivo2> ... <archivoN>

Donde <archivo1> a <archivoN> son los archivos a mostrar.

Ejemplo: “less articulos/historia” muestra por la pantalla el contenido del archivo “historia” dentro del directorio “articulos”.

man

Muestra la página de manual del comando o recurso (función de librería, archivo de configuración) dado.

Sintaxis: man <comando>

Donde <comando> es el nombre del comando o recurso sobre el que queremos obtener información.

Ejemplo: “man ls” muestra ayuda sobre el comando “ls”.

grep

Muestra todas las líneas de un archivo dado que coinciden con un cierto patrón.

Sintaxis: grep <patrón> <archivo1> <archivo2> ... <archivoN>

Donde <patrón> es una expresión regular[19] y <archivo1> a <archivoN> son los archivos donde buscar.

Ejemplo: “grep micasa /etc/hosts” mostrará todas las líneas en el archivo /etc/hosts que contienen la cadena “micasa”.

6. Caracteres “comodín”

Una característica importante de la mayoría de los intérpretes de comandos en *Unix* es la capacidad para referirse a más de un archivo usando *expresiones regulares*. En su forma más simple, esto consiste en la utilización de los llamados “comodines” que permiten referirse a, por ejemplo, todos los archivos cuyo nombre contiene una determinada secuencia de caracteres.

[19] Las *expresiones regulares* son una forma de describir un conjunto de cadenas.

6.1. Los comodines “*” y “?”

El comodín “*” hace referencia cualquier cadena de caracteres en el nombre del archivo. Cuando se usa el carácter “*” para referirse al nombre de un archivo, el intérprete de comandos lo sustituye por todas las combinaciones posibles provenientes de los archivos en el directorio al cual nos estamos refiriendo.

Veamos un ejemplo rápido. Supongamos que `diego` tiene los archivos `hugo`, `paco` y `luis` en el directorio actual.

```
/home/diego$ ls
hugo luis paco
/home/diego$
```

Para listar todos los archivos con la letra “o” en su nombre, usamos la orden:

```
/home/diego$ ls *o*
hugo paco
/home/diego$
```

Como podemos ver, el comodín “*” ha sido sustituido con todas las combinaciones posibles que coincidían de entre los archivos del directorio actual.

El uso de “*” solo, hace referencia a todos los archivos, puesto que todos los caracteres coinciden con el comodín.

```
/home/diego$ ls *
hugo luis paco
/home/diego$
```

Veamos algunos otros ejemplos:

```
/home/diego$ ls h*
hugo
/home/diego$ ls *is
luis
/home/diego$ ls *u*
hugo luis
/home/diego$ ls p*o
paco
/home/diego$
```

El proceso de la sustitución de caracteres como “*” en nombres de archivos es llamado “*expansión de comodines*” y es efectuado por el intérprete de comandos. Esto es importante: los comandos, como `ls`, nunca ven el “*” en su lista de parámetros. Es el intérprete quien expande los comodines para incluir todos los nombres de archivos que se adaptan. Por lo tanto la orden:

```
/home/diego$ ls *o*
```

es expandida para obtener

```
/home/diego$ ls hugo paco
```

Otro carácter comodín es "?". Este carácter comodín sólo expande un único carácter. Luego "ls ?" mostrará todos los nombres de archivos con un carácter de longitud, y "ls termca?", por ejemplo, mostrará "termcap" pero no "termcap.backup". Aquí tenemos otro ejemplo:

```
/home/diego$ ls hu?o
hugo
/home/diego$ ls p??o
paco
/home/diego$ ls ???s
luis
/home/diego$
```

Como podemos ver, los caracteres comodín nos permiten referirse a más de un archivo a la vez. En el sumario de comandos básicos dijimos que cp y mv pueden copiar o mover múltiples archivos de una vez. Por ejemplo:

```
/home/diego$ cp /etc/s* /home/diego
```

copiará todos los archivos de /etc que comiencen con "s" al directorio /home/diego.

6.2. Los comodines y los archivos ocultos

Los comodines "*" y "?" no coincidirán con nombres de archivos que comiencen con un punto ("."). Estos archivos son tratados como *ocultos*. Los archivos de este tipo no son mostrados en un listado normal de ls y no son afectados por el uso de "*" y "?".

Por ejemplo, ya hemos mencionado que cada directorio tiene dos entradas especiales: ".", que hace referencia al directorio actual, y "..", que se refiere al directorio padre. Como ya hemos visto, al usar ls esas dos entradas no se muestran:

```
/home/diego$ ls
hugo luis paco
/home/diego$
```

Si usamos el parámetro "-a" con ls podremos ver nombres de archivos que comienzan con ".". Observemos:

```
/home/diego$ ls -a
.  ..  .bash_profile .bashrc hugo luis paco
/home/diego$
```

Ahora podemos ver las dos entradas especiales, "." y "..", así como otros dos archivos ocultos: ".bash_profile" y ".bashrc".[20]

7. Comunicación entre procesos

Unix provee una serie de mecanismos para poder intercomunicar procesos[21], como parte de la estrategia de combinar el uso de herramientas simples para poder resolver problemas complejos.

[20]Estos dos archivos son usados en el arranque por el intérprete de comandos bash cuando diego realiza el login.
[21]Se denomina proceso a un programa en ejecución

7.1. Entrada y salida estándar

La mayoría de los programas de *Unix* toman (leen) sus datos de entrada de la llamada "entrada estándar" y envían sus resultados (escriben) a la "salida estándar" (a menudo abreviadas como "`stdin`" y "`stdout`" respectivamente). Usualmente el sistema está configurado de forma que la entrada estándar es el teclado y la salida estándar la pantalla (recordemos que al teclado y la pantalla se los denomina comunmente "consola").

Veamos un ejemplo con el comando `cat`. Normalmente `cat` lee datos de los archivos cuyos nombres se pasan como argumentos en la línea de comandos y envía estos datos directamente a la salida estándar. Luego, al ejecutar la orden:

```
/home/diego/articulos$ cat historia tesis
```

se mostrará por pantalla el contenido del archivo `historia` seguido por el contenido del archivo `tesis`.

Si `cat` no recibe nombres de archivos como parámetros, leerá datos de `stdin` y los enviará a `stdout`. Veamos un ejemplo:

```
/home/diego/articulos$ cat
Hola !!!
Hola !!!
Adiós.
Adiós.
[Ctrl]+[D]
/home/diego/articulos$
```

Como puede verse, cada línea que el usuario teclea (en letra itálica) es inmediatamente reenviada al monitor por `cat`. Cuando están leyendo datos, los procesos reconocen el fin de la entrada al recibir el carácter `EOT` ("*end-of-text*", fin de texto). Normalmente es generado con la combinación de teclas `[Ctrl]+[D]`.

Veamos otro ejemplo. El comando `sort` toma como entrada líneas de texto (al igual que `cat`, leerá desde `stdin` si no se le proporcionan nombres de archivos en la línea de comandos), y devuelve la salida ordenada por `stdout`. Probemos lo siguiente:

```
/home/diego/articulos$ sort
bananas
manzanas
duraznos
[Ctrl]+[D]
bananas
duraznos
manzanas
/home/diego/articulos$
```

7.2. Redirigiendo la entrada y salida

Ahora, supongamos que queremos que la salida de `sort` vaya a un archivo llamado `compras` en vez de a la pantalla. El intérprete de comandos nos permite redirigir la salida estándar a un archivo usando el símbolo ">". Veamos como funciona:

```
/home/diego/articulos$ sort > compras
bananas
```

```
manzanas
duraznos
[Ctrl]+[D]
/home/diego/articulos$
```

Como podemos ver, el resultado de `sort` no se muestra por pantalla, en su lugar es escrito en el archivo `compras`. Veamos ahora su contenido:

```
/home/diego/articulos$ cat compras
bananas
duraznos
manzanas
/home/diego/articulos$
```

Supongamos ahora que tenemos nuestra lista desordenada original en el archivo `cosas`. Una forma de ordenar la información y escribirla en un archivo podría ser darle a `sort` el nombre del archivo a leer en lugar de la entrada estándar y redirigir la salida estándar como acabamos de hacer:

```
/home/diego/articulos$ sort cosas > compras
/home/diego/articulos$ cat compras
bananas
duraznos
manzanas
/home/diego/articulos$
```

Hay otra forma de hacer esto. No sólo puede ser redirigida la salida estándar, también puede ser redirigida la entrada estándar usando el símbolo “<”.

```
/home/diego/articulos$ sort < cosas
bananas
duraznos
manzanas
/home/diego/articulos$
```

Las órdenes “`sort < cosas`” y “`sort cosas`” tienen el mismo efecto, pero esto nos permite ver que el intérprete de comandos es quien maneja las redirecciones. `sort` no recibe el nombre del archivo a leer, desde su punto de vista, está leyendo datos de la entrada estándar como si fueran ingresados desde el teclado.

Esto introduce el concepto de “filtro”. Un filtro es un programa que lee datos de la entrada estándar, los procesa de alguna forma y devuelve el resultado por la salida estándar. Usando la redirección, tanto la entrada estándar como la salida estándar pueden ser radirigidas a archivos. `sort` es un filtro simple: ordena los datos de entrada y envía el resultado a la salida estándar. `cat` es incluso más simple, no hace nada con los datos de entrada, simplemente los envía a la salida tal como los recibe.

7.3. Redirección no destructiva

El uso de “>” para redirigir la salida a un archivo es destructivo. En otras palabras, la orden:

```
/home/diego/articulos$ ls > listado
```

sobreescribe el contenido del archivo `listado`. Si en su lugar, usamos los símbolos ">>", la salida será añadida al final del archivo nombrado, en lugar de ser sobrescrito (creándolo, si este no existiera). El comando:

```
/home/diego/articulos$ ls >> listado
```

añadirá la salida de `ls` al final de `listado`.

7.4. Uso de tuberías (pipes)

Ya hemos visto como usar `sort` como un filtro, pero estos ejemplos suponen que tenemos los datos en un archivo o que los introducimos manualmente por la entrada estándar.

¿Qué pasa si los datos que queremos ordenar provienen de la salida de otro comando, como `ls`? Por ejemplo, si el contenido de nuestro directorio actual fuese:

```
/home/diego/articulos$ ls
historia
ingles
notas
tesis
/home/diego/articulos$
```

Usando la opción "`-r`" con `sort` ordenaremos los datos en orden inverso. Una forma de hacer esto sería:

```
/home/diego/articulos$ ls > listado
/home/diego/articulos$ sort -r listado
tesis
notas
ingles
historia
/home/diego/articulos$
```

Aquí, escribimos la salida de `ls` en un archivo y luego ejecutamos "`sort -r`" sobre él. Pero de esta forma hemos creado un archivo solamente para que los datos generados por `ls` luego puedan ser leídos por `sort`. No parece tener demasiado sentido hacer esto.

La solución a este problema es usar los "pipes" (o "tuberías"). Los pipes son una poderosa herramienta provista por el sistema para conectar dos procesos, de manera que la `stdout` del primero es enviada directamente a la `stdin` del segundo (esto puede generalizarse para formar una cadena de procesos). Para crear un pipe se usa el símbolo "|". En nuestro ejemplo, queremos conectar la salida de `ls` con la entrada de `sort`:

```
/home/diego/articulos$ ls | sort -r
tesis
notas
ingles
historia
/home/diego/articulos$
```

Esta forma es más corta, más eficiente y hasta más fácil de escribir.

Veamos otro ejemplo útil. Al usar el comando:

```
/home/diego/articulos$ ls /usr/bin
```

se mostrará una lista de archivos demasiado extensa, parte de la cual pasará rápidamente por la pantalla ante nuestros ojos, sin que podamos leerla. En lugar de esto, usemos `less` para detener el listado cada vez que se complete la pantalla:[22]

```
/home/diego/articulos$ ls /usr/bin | less
```

Ahora podemos ir avanzando o retrocediendo línea por línea o pantalla por pantalla, cómodamente.

Como dijimos anteriormente, podemos "entubar" más de dos procesos a la vez. `head` es un filtro que muestra las primeras líneas de la entrada. Si queremos ver el último archivo del directorio actual en orden alfabético, usaremos:

```
/home/diego/articulos$ ls | sort -r | head -1
tesis
/home/diego/articulos$
```

Donde "`head -1`" muestra la primera línea de la entrada que recibe (en este caso, el flujo de datos ordenados inversamente, proveniente de "`sort -r`", que es el listado que éste recibió de "`ls`").

8. Permisos de archivos

Al ser *Unix* un sistema multiusuario, los archivos de cada usuario deben ser protegidos del resto de los usuarios. Lo mismo ocurre con los archivos del sistema (programas, configuraciones, etc.). Esto tiene que ver no sólo con la confidencialidad de la información, sino también con la protección de errores involuntarios por parte de los usuarios. Para ello se utiliza un sistema de "permisos de archivos". Este mecanismo permite que archivos y directorios "pertenezcan" a un usuario en particular. Por ejemplo, como `diego` creó archivos en su directorio "home", diego es el propietario de esos archivos y tiene acceso total a ellos.

Unix también permite que los archivos sean compartidos entre usuarios y grupos de usuarios. Si `diego` lo desea, podría restringir el acceso a sus archivos de forma que ningún otro usuario pueda acceder a ellos.

8.1. Tipos de permisos

Cada archivo pertenece a un usuario y a un grupo en particular. Un grupo es un conjunto de usuarios definido (cada usuario pertenece al menos a un grupo, pero puede pertenecer a varios).

Los grupos usualmente son definidos por el tipo de usuarios que acceden al sistema. Por ejemplo, en un sistema *Unix* de una universidad, los usuarios pueden ser divididos en los grupos `estudiantes`, `dirección`, `profesores` e `invitados`. Hay también unos pocos grupos definidos por el sistema (como `bin` y `daemon`) que son usados por el propio sistema para controlar el acceso a los recursos. Normalmente los usuarios comunes no pertenecen a estos grupos.

Los permisos están divididos en tres tipos: lectura, escritura y ejecución. Estos permisos pueden ser fijados para tres clases de usuarios: el propietario del archivo o directorio, los integrantes del grupo al que pertenece y todos los demás usuarios.

El permiso de lectura permite a un usuario leer el contenido del archivo o en el caso de un directorio, listar el contenido del mismo (usando `ls`).

[22]Recordemos que para salir de `less` hay que presionar `[Q]`.

El permiso de escritura permite a un usuario escribir y modificar el archivo (inclusive, eliminarlo). Para directorios, el permiso de escritura permite crear nuevos archivos o borrar archivos ya existentes en el mismo.

Por último, el permiso de ejecución permite a un usuario ejecutar el archivo si es un programa. Para directorios, el permiso de ejecución permite al usuario ingresar al mismo (por ejemplo, con el comando `cd`).

8.2. Interpretando los permisos de archivos

Veamos un ejemplo del uso de permisos de archivos. Usando el comando `ls` con la opción "`-l`" se mostrara un listado "largo" de los archivos, el cual incluye los permisos.

```
/home/diego/nuevo$ ls -l cosas
-rw-r-r- 1 diego users 505 Mar 13 19:05 paco
/home/diego/nuevo$
```

El primer campo representa los permisos del archivo. El tercer campo es el propietario del mismo (`diego`), el cuarto es el grupo al cual pertenece el archivo (`users`) y el último campo es el nombre del archivo (`paco`).

La cadena "`-rw-r-r-`" nos informa, por orden, los permisos para el propietario, los usuarios del grupo y el resto de los usuarios.

El primer carácter de la cadena de permisos ("`-`") representa el tipo de archivo. El "`-`" significa que es un archivo regular, "`d`" indicaría que se trata de un directorio. Los siguientes tres caracteres ("`rw-`") representan los permisos para el propietario del archivo, `diego`. Éste tiene permisos para leer (`r`) y escribir (`w`) en el archivo `paco`.

Como ya mencionamos, además de los permisos de lectura y escritura existe el permiso de ejecución, representado por una "`x`". Como hay un "`-`" en lugar de la "`x`", significa que `diego` no tiene permiso para ejecutar ese archivo. Esto es correcto, puesto que `paco` no es un programa. Por supuesto, como el archivo es de `diego`, él puede cambiar los permisos, dándose a sí mismo permiso de ejecución, como veremos más adelante.

Los siguientes tres caracteres, "`r-`", representan los permisos para los miembros del grupo al que pertenece el archivo (en este caso, `users`). Como sólo aparece una "`r`" cualquier usuario que pertenezca al grupo `users` puede leer este archivo, pero no modificarlo ni ejecutarlo.

Los últimos tres caracteres, "`r-`", representan los permisos para cualquier otro usuario del sistema (que no sea `diego` ni pertenezca al grupo `users`). Nuevamente, como sólo está presente la "`r`", los demás usuarios pueden leer el archivo, pero no escribir en él o ejecutarlo.

Aquí tenemos otros ejemplos de permisos de grupo.

`-rwxr-xr-x`

El propietario del archivo puede leer, escribir y ejecutar el archivo. Los usuarios pertenecientes al grupo del archivo y todos los demás usuarios pueden leer y ejecutar el archivo.

`-rw----`

El propietario del archivo puede leer y escribir. Nadie más puede acceder al archivo.

`-rwxrwxrwx`

Todos los usuarios pueden leer, escribir y ejecutar el archivo.

`drwxr-xr-x`

El propietario del directorio puede leer, escribir y entrar al mismo. Los usuarios pertenecientes al grupo del directorio y todos los demás usuarios pueden leer e ingresar al directorio.

8.3. Dependencias

Es importante remarcar que los permisos de un archivo también dependen de los permisos del directorio en el que reside. Por ejemplo, aunque un archivo tenga los permisos "`-rwxrwxrwx`", otros usuarios no podrán acceder a él a menos que también tengan permiso de lectura y ejecución para el directorio en el cual se encuentra el archivo. Si `diego` quiere restringir el acceso a todos sus archivos, podría simplemente poner los permisos de su directorio "home", `/home/diego`, como "`drwx---`". De esta forma ningún usuario podrá acceder a su directorio y, por lo tanto, tampoco a ninguno de sus archivos o subdirectorios. Así `diego` no necesita preocuparse de los permisos individuales de cada uno de sus archivos.

En otras palabras un usuario, para acceder a un archivo, debe de tener permiso de ejecución de todos los directorios a lo largo del camino de acceso al archivo, además de permiso de lectura del archivo en particular.

La mayoría de los archivos usualmente tienen permisos "`-rw-r-r-`", lo que permite a todos los usuarios leer los archivos, pero solamente a su propietario modificarlos. Los directorios usualmente tienen los permisos "`drwxr-xr-x`", lo que permite a todos lo susuarios moverse y ver los directorios, pero solo permiten a su propietario poder crear o borrar archivos en ellos.

Si un usuario desea limitar el acceso de otros a un archivo en particular, puede asignarle los permisos "`-rw----`. De la misma manera, poniendo los permisos de un directorio como "`drwx---`" no se permitirá el acceso a los demás usuarios.

8.4. Cambiando permisos

El comando `chmod` se usa para establecer los permisos de un archivo. Sólo el propietario puede cambiar los permisos del archivo (además, claro está, del administrador del sistema, el usuario `root`). La sintaxis de `chmod` es:

```
chmod [a,u,g,o][+,-][r,w,x] <archivos>
```

El primer parámetro indica a qué usuarios afecta: *all*, *user*, *group* u *other* (todos, el propietario, el grupo u otros usuarios; respectivamente). Luego se especifica si se están añadiendo permisos (+) o quitándolos (-). El tercer parámetro especifica qué tipo de permiso estamos añadiendo o quitando: *read*, *write* o *execute*. Finalmente, se indican los nombres de los archivos a afectar. Algunos ejemplos de la utilización de `chmod` son:

`chmod a+r paco`

Da a todos los usuarios acceso de lectura al archivo `paco`.

`chmod +r paco`

Igual al anterior. Si no se indica `a`, `u`, `g` u `o` por defecto se toma `a`.

`chmod og-x paco`

Quita permisos de ejecución de `paco` a todos los usuarios excepto al propietario.

`chmod u+rwx paco`

Permite al propietario leer, escribir y ejecutar el archivo `paco`.

`chmod o-rwx paco`

Quita permisos de lectura, escritura y ejecución del archivo `paco` a todos los usuarios menos al propietario y a los usuarios del grupo.

`chmod og-r paco luis`

Quita permisos de lectura de los archivos `paco` y `luis` a todos los usuarios excepto al propietario.

9. Próximos pasos

Luego de recorrer en este breve tutorial los conceptos básicos del sistema *GNU/Linux* y habiendo adquirido las habilidades necesarias para comenzar a trabajar en él, usted se encuentra en condiciones de abordar otros temas. A modo de sugerencia, van los siguientes.

9.1. Editores de textos

En un entorno *Unix* se hace indispensable el uso de un editor de textos. La variedad de opciones disponibles es inmensa: desde editores extremadamente simples, hasta algunos muy completos y complejos.

La lista a continuación enumera algunos de los más conocidos:

nano

Editor muy fácil de usar. No posee capacidades avanzadas, pero es simple de usar sin conocimientos previos.

joe

Editor similar al *WordStar* de *DOS*. De dificultad y potencia medias.

vim

Versión mejorada del clásico de *Unix* `vi`. Requiere cierto entrenamiento inicial para su uso, pero posibilita lograr una gran productividad en la edición de textos. Viene acompañado de un tutorial interactivo llamado `vimtutor`.

emacs

El editor estrella del proyecto *GNU*. Ofrece una gran cantidad de herramientas y es mucho más que un editor de textos, aunque requiere cierto entrenamiento en su uso.

A modo de sugerencia, tenga en cuenta que todo el tiempo que invierta en el aprendizaje de un buen editor de textos (como `emacs` o `vim`) redundará en una mayor productividad en el uso del sistema.

9.2. Programación del shell

Los intérpretes de comando ofrecen potentes mecanismos para poder resolver problemas de gran complejidad. Entre ellos, `bash` es el más utilizado actualmente, existiendo una amplia documentación sobre su uso (desde "`man bash`" hasta los *HOWTO* disponibles en el "*Linux Documentation Project*".[23]

Se denomina "*shell script*" a un archivo que contenga comandos del shell. Mediante los shell scrips, podemos escribir verdaderos programas que automaticen tareas de mantenimiento del sistema o que realicen otras tareas comunes.

9.3. Administración del sistema

Una vez adquirido cierto dominio sobre el entorno *GNU/Linux* estará listo para adentrarse en las tareas reservadas al usuario `root`. Entre ellas se cuentan la administración de usuarios, la instalación de paquetes de software, la realización de copias de respaldo, entre tantas otras.

Nuevamente, en el "*Linux Documentation Project*" encontrará una amplia documentación sobre estos temas.

[23] *http://www.tldp.org/*

9.4. Distribuciones de *GNU/Linux*

Existen distintas distribuciones de *GNU/Linux*, orientadas a distinto tipo de usuarios o de sistemas, con herramientas de administración diferentes, o con distintas políticas. Algunas son comercializadas por empresas, en tanto que otras son mantenidas por comunidades de usuarios particulares. Algunas son gratuitas, son pagas. Unas requieren de horas (e incluso días) de instalación, en tanto que otras permiten tener un sistema 100 % funcional en cuestión de minutos (¡hasta hay algunas que no requieren instalación en el disco duro!).

Sin duda encontrará la más apropiada para usted (o hasta pueden ser varias), luego de experimentar con varias de ellas, evaluando sus pro y sus contras. A modo de orientación, estas son las más populares en la actualidad:

RedHat

Es la distribución preferida por las empresas por su excelente soporte comercial. Existe una versión gratuita, orientada a la comunidad, llamada *Fedora*.

Debian

Es una distribución mantenida por un grupo de colaboradores que incluye principalmente *software libre*.

Ubuntu

Es una distribución derivada de *Debian*, con soporte comercial de la empresa que la produce. Puede ejecutarse sin necesidad de instalación en el disco duro.

Gentoo

Es una distribución de instalación lenta, pero que permite optimizar todo el software al hardware específico utilizado.

Mandriva

Distribución derivada de *RedHat* (antes llamada "*Mandrake Linux*") que muchos usuarios prefieren por su simplicidad de instalación.

SuSE

Distribución de origen alemán, fácil de instalar y administrar, adquirida por la empresa *Novell*.