Preparación para examen de certificación 101 LPI (segundo lanzamiento) Parte I

Índice

1. Sección 1. Antes de Comenzar

Unas palabras sobre este Tutorial

Bienvenido a "Fundamentos de Linux", el primero de cuatro tutoriales diseñado para prepararlo para el examen 101 de *Linux Professional Institute* (Instituto Profesional de Linux). En este tutorial, haremos una introducción a bash (el shell estándar de linux), veremos como tomar ventajas de los comandos Linux estándar (como ls, cp o mv), explicaremos i-nodos y enlaces simbólicos y permanentes (*soft y hard links*), y mucho más. Al final de este tutorial, usted contará con una sólida base en los fundamentos de Linux, y estará listo para aprender algunas tareas básicas de administración de su sistema Linux. Cuando finalice esta serie de tutoriales (ocho en total) usted tendrá el conocimiento necesario para ser un Administrador de Sistemas Linux, y estará preparado para rendir (si así lo quisiera) para la certificación LPIC de nivel 1 del Linux Professional Institute.

Este tutorial en particular (Parte I) es ideal para aquellos que son nuevos en Linux, o para aquellos que quieran revisar o incrementar sus conocimientos de los conceptos fundamentales de Linux como copiar y mover archivos, crear enlaces simbólicos y permanentes, o usar los comandos estándar de Linux para procesamiento de textos junto con tuberías (*pipelines*) y redirecciones. A lo largo del camino aparecerán consejos, trucos y sugerencias para mantener el tutorial interesante y práctico, aún para aquellos con una buena experiencia en Linux. Para los principiantes este material resultará nuevo, pero los más experimentados usuarios de Linux verán que este tutorial completará y redondeará su manejo de los fundamentos de Linux.

Acerca del autor

Residiendo en Albuquerque, Nuevo México, Daniel Robbins es el Arquitecto en jefe de la metadistribución Gentoo Linux. También escribe artículos y seminarios para IBM developerWorks y Servicios de Desarrollo de Intel, y ha contribuído en la edición de distintos libros como Samba y Linux SuSE. Daniel disfruta su tiempo con su esposa Mary y su hija Hadassah. Puede contactarlo en la siguiente dirección: drobbins@gentoo.org.
Para consultas técnicas sobre el contenido de éste seminario, contáctese con el autor, a drobbins@gentoo.org.

Unas palabras sobre la Traducción

En el Departamento Universitario de Informática (DUI) de la Universidad Nacional de Córdoba, en la provincia de Córdoba, Argentina; se tradujo este seminario del idioma inglés al español para el habla hispana. El trabajo de traducción fue desarrollado por las siguientes personas:
Autor: Daniel Robbins
Coordinación general de la Traducción: José Daniel Britos
Traducción y compaginación: Carlos Alejandro Julian
Colaboración: Ana Gabriela Llimós y Javier Mansilla

Si bien hemos tratado de traducir completamente este texto de manera tal que una persona no anglo parlante logre tener un total conocimiento del contenido de este tutorial, también hemos dejado ciertos términos en su idioma original (inglés) por ser palabras con las que usted se encontrará frecuentemente al utilizar su sistema Linux, de manera tal que comience a familiarizarse con ellas.

2. Sección 2. Introducción a bash

Un shell

Si usted ha usado un sistema Linux, sabrá que cuando usted ingresa al sistema (*log in*) es saludado con un prompt que luce más o menos así:

```
$
```

El prompt que usted realmente vea en su sistema puede lucir un tanto diferente. Puede contener el nombre de host de su sistema *hostname*, el nombre de directorio actual de trabajo, ambos, etc. Más allá de como se vea su prompt lo importante a saber lo siguiente: el programa que imprimió ese prompt en su pantalla se llama "shell" (cáscara en español), y es muy probable que su shell particular sea uno llamado bash.

¿Está usted usando bash?

Puede revisar si usted está utilizando bash tipeando:

```
$ echo $SHELL
/bin/bash
```

Si con la linea de arriba obtuvo un mensaje de error o no respondió de la misma forma que en nuestro ejemplo, entonces usted está usando algún otro shell, pero si usted se está preparando para el examen 101, le recomendamos cambiar a bash. (El próximo tutorial, sobre administración básica, cubre cómo cambiar su shell a través del comando chsh).

Sobre bash

Bash, es un acrónimo para *Bourne-again-shell* (shell re-nacido), es el shell por defecto en la mayoría de los sistemas Linux. El trabajo de un shell es obedecer sus comandos de manera que usted pueda interactuar con su sistema Linux. Cuando termine de introducir comandos, puede solicitarle a su shell salir a través de exit o logout. En ese momento usted regresará al prompt de inicio de sesión (log in).

A propósito, también puede salir de su sesión tecleando control-D en prompt de su bash

Usando el comando cd

Como seguramente habrá notado, situarse en prompt de su bash no es para nada emocionante. Así que comencemos a usar bash para navegar alrededor de nuestro sistema de archivos *filesystem*. En el prompt, tipee lo siguiente (obviamente sin incluir el $):

```
$ cd /
```

Con esto usted le ha dicho a bash que usted quiere trabajar en el directorio /, también conocido como *root* (raíz). Todos los directorios en el sistema forman un árbol, y / es considerado la raíz de éste. cd ajusta al directorio donde usted trabajará, también conocido como *"current working directory"* (actual directorio de trabajo).

Rutas (*Paths*)

Para ver el directorio actual de trabajo de bash, escriba:

```
$ pwd
/
```

En el ejemplo anterior (sobre cd), el argumento / es llamado un *path* (como ya dijimos, un camino o ruta) hacia donde queremos dirigirnos. En particular, el argumento / es un *path* absoluto, o sea que la ubicación dentro del árbol del sistema de archivos es indicada en forma relativa a su *root*.

Rutas Absolutas

Como muestra, acá hay varias rutas absolutas distintas a las del ejemplo anterior:

```
/dev
/usr
/usr/bin
/usr/local/bin
```

Como podrá ver, lo que estas rutas absolutas tienen en común es que todas comienzan con una barra /. Si le damos a cd el path /usr/local/bin, primero cd entrará al directorio /, desde allí luego entrará al directorio usr, y recién desde allí entrará a bin. Las rutas absolutas siempre comienzan a evaluarse a partir de /.

Rutas Relativas

El otro tipo de rutas es el de las rutas relativas. bash, cd, y otros comandos siempre interpretan este tipo de rutas como relativas al directorio actual de trabajo. Las rutas relativas nunca comienzan con una barra /. Así que si estamos en /usr :

```
$ cd /usr
```

Entonces, ahora usted puede usar una ruta relativa para cambiar el directorio actual a /usr/local/bin de la siguiente forma:

```
$ cd local/bin
$ pwd
/usr/local/bin
```

Usando el directorio .. (punto punto)

Las rutas relativas pueden también contener uno o mas directorios .. (punto punto). El directorio .. es un directorio especial que apunta al directorio padre del actual. De esta manera, y continuando con el ejemplo anterior:

```
$ pwd
/usr/local/bin
$ cd ..
$ pwd
/usr/local
```

Como usted puede ver, el nuevo directorio actual de trabajo es /usr/local. De esta forma podemos "volver atrás" un directorio con respecto al cual estamos situados.

Más usos de los .. (punto punto)

Además, podemos incluir el directorio .. en alguna ruta relativa, permitiéndonos dirigirnos a algún directorio que se encuentre en alguna rama lateral del árbol de directorios (esto es, obviamente, sin tener que utilizar una ruta absoluta). Por ejemplo:

```
$ pwd
/usr/local
```

```
$ cd ../share
$ pwd
/usr/share
```

Ejemplos de rutas relativas

Las rutas relativas pueden volverse un tanto complicadas. Acá van algunos ejemplos, ninguno muestra el directorio al cual se accedió. Intente entender donde quedará situado después de tipear cada uno de estos comandos:

```
$ cd /bin
$ cd ../usr/share/zoneinfo

$ cd /usr/X11R6/bin
$ cd ../lib/X11

$ cd /usr/bin
$ cd ../bin/../bin
```

Ahora, pruebe chequear lo que usted pensó con lo que realmente sucede al teclear los comandos.

Entendiendo . (un punto)

Antes de terminar con el comando cd hay algunas cosas más que vale la pena mencionar. Primero hablaremos de otro directorio especial llamado . (un punto), que se refiere al directorio actual. Aunque este directorio no suele ser utilizado con el comando cd, si es útil, por ejemplo, para ejecutar algún programa situado en el directorio actual, como se ve:

```
$ ./miprograma
```

En este ejemplo, se ejecutará el archivo ejecutable miprograma que resida en el directorio actual de trabajo.

---

cd y el directorio *home*

El directorio *home* es, para cada usuario, el directorio sobre el cual puede trabajar libremente. Si lo que usted quiere es cambiar el directorio actual a su directorio *home*, entonces escriba:

```
$ cd
```

Cuando utilice el comando cd sin argumentos, este lo llevara a su directorio *home* (casa en castellano). Será el directorio /root para el super-usuario, y /home/nombredeusuario para usuarios comunes. Ahora bien... ¿qué sucede si lo que usted quiere es referirse a un archivo situado en su directorio home? Quizás lo que usted quiera sea pasar un archivo como argumento a su comando miprograma. Si el archivo está es su directorio home, entonces usted puede escribir:

```
$ ./miprograma /home/juanlopez/miarchivo.txt
```

Sin embargo, usar rutas absolutas no es siempre lo más conveniente ni cómodo. Afortunadamente, puede hacer uso del carácter (tilde) para hacer lo mismo:

```
$ ./miprograma ~/miarchivo.txt
```

Directorios *home* de otros usuarios

bash expandirá un simple ~ (tilde) para apuntar a su propio directorio home, pero también usted puede usar el tilde para apuntar a los directorio home de los otros usuarios. Por ejemplo, si quiere referirse a un archivo llamado lucas.txt en el directorio home de lucasperez, entonces puede tipear lo siguiente:

```
$ ./miprograma ~lucasperez/lucas.txt
```

3. Sección 3. Usando los comandos de Linux

Introduciendo el comando ls

Ahora, vamos a darle una pequeña mirada al comando ls. Muy probablemente usted ya haya tenido contacto con el comando ls, pero por si no lo sabe, este sirve para listar el contenido del directorio actual de trabajo:

```
$ cd /usr
$ ls
X11R6       doc          i686-pc-linux-gnu   lib       man           sbin    ssl
bin         gentoo-x86   include             libexec   portage       share   tmp
distfiles   i686-linux   info                local     portage.old   src
```

Especificando la opción -a, podrá ver todos los archivos del directorio, incluyendo los archivos ocultos (aquellos que comienzan con un punto (.). Como se ve en el siguiente ejemplo, ls -a muestra también los directorios especiales . y .. (en realidad son enlaces a esos dos directorios):

```
$ ls
.       bin          gentoo-x86          include   libexec   portageshare   tmp
..      distfiles    i686-linux          info      local     portage.old    src
X11R6   doc          i686-pc-linux-gnu   lib       man       sbin           ssl
```

Listado completo de información

Al comando ls usted le puede especificar uno o más archivos o directorios a través de la linea de comandos. Si especifica un archivo, ls sólo mostrará este archivo, pero si especifica un directorio, entonces ls listará el contenido de ese directorio.La opción -l le resultará de gran utilidad cuando quiera ver información sobre permisos, tiempos de modificación, tamaño o propiedad de los contenidos listados.

Listado completo de información, continuamos

En el siguiente ejemplo, se utiliza la opción -l para ver una listado del directorio /usr, pero con información completa:

```
$ ls -l /usr
drwxr-xr-x    7 root   root       168 Nov 24 14:02 X11R6
drwxr-xr-x    2 root   root     14576 Dec 27 08:56 bin
drwxr-xr-x    2 root   root      8858 Dec 26 12:22 distfiles
lrwxrwxrwx    1 root   root         9 Dec 22 20:07 doc -> share/doc
```

```
drwxr-xr-x   62 root   root      1856 Dec 27 09:23 gentoo-x86
drwxr-xr-x    4 root   root       152 Dec 12 23:10 i686-linux
drwxr-xr-x    4 root   root        96 Nov 24 13:17 i686-pc-linux-gnu
drwxr-xr-x   54 root   root      5992 Dec 24 22:30 include
lrwxrwxrwx    1 root   root        10 Dec 22 20:43 info -> share/info
drwxr-xr-x   28 root   root     13552 Dec 26 00:31 lib
drwxr-xr-x    3 root   root        72 Nov 25 00:34 libexec
drwxr-xr-x    8 root   root       240 Dec 22 20:21 local
lrwxrwxrwx    1 root   root         9 Dec 22 20:21 man -> share/man
lrwxrwxrwx    1 root   root        11 Dec  8 07:59 portage -> gentoo-x86/
drwxr-xr-x   60 root   root      1864 Dec  8 07:55 portage.old
drwxr-xr-x    3 root   root      3098 Dec 22 14:35 sbin
drwxr-xr-x   46 root   root      1144 Dec 24 15:43 share
drwxr-xr-x    8 root   root       328 Dec 26 00:07 src
drwxr-xr-x    6 root   root       176 Nov 24 14:25 ssl
lrwxrwxrwx    1 root   root        10 Dec 22 20:57 tmp -> ../var/tmp
```

La primer columna muestra la información sobre permisos para cada ítem listado. Más adelante explicaremos como interpretar esta información. La columna siguiente lista el número de links para cada objeto del *filesystem*. También esto será explicado a la brevedad. La tercer y cuarta columna listan el propietario del elemento, y el grupo al cual pertenece, respectivamente. La quinta muestra el tamaño de los objetos, mientras que la sexta lista cuando fue realizada la última modificación del objeto (*"last modified time"* o *"mtime"*). La última columna es el nombre del objeto. Si el archivo es un enlace simbólico, entonces usted verá una flechita -> y la ruta hacia la cual el link simbólico apunta.

Mirando los directorios

A veces, usted querrá mirar *los* directorios, en vez de dentro de ellos. Para estas situaciones puede especificar la opción -d, la que le dirá a ls que mire los directorios, y no dentro de ellos como normalmente sucede:

```
$ ls -dl /usr /usr/bin /usr/X11R6/bin ../share
drwxr-xr-x    4 root   root       152 Dec 12 23:10 ../share
drwxr-xr-x   17 root   root       576 Dec 12 07:45 /usr
drwxr-xr-x    2 root   root      3192 Dec 12 14:13/usr/X11R6/bin
drwxr-xr-x    2 root   root     14576 Dec 12 20:08 /usr/bin
```

Listados recursivos y de inodos

Puede usar ls -d para mirar los directorios sin desreferenciarlos, o por el contrario puede utilizar -R para hacer lo opuesto: no sólo mirar dentro de los directorios, sino que recursivamente mirar dentro de todos los directorios dentro del actual. No incluiremos ningún ejemplo de la salida que produce esta opción (ya que generalmente es muy grande), pero usted puede probar con los comandos ls -R y ls -Rl para tener una idea de como trabajan. Finalmente, la opción -i para el comando ls puede ser usada para mostrar el número de inodos (*inodes*) de los objetos del *filesystem* listados:

```
$ ls -i
1409 X11R6        314258 i686-linux          43090 libexec      13394 sbin
 1417 bin           1513 i686-pc-linux-gnu    5120 local        13408 share
 8316 distfiles     1517 include               776 man          23779 src
   43 doc           1386 info                93892 portage      36737 ssl
70744 gentoo-x86    1585 lib                  5132 portage.old     784 tmp
```

Entendiendo inodos, primera parte

En un sistema de archivos, a cada objeto le es asignado un índice único, llamado un número de *inodo*. Esto parecería ser trivial, pero entender completamente el concepto de inodos es esencial para comprender muchas de las operaciones de un *filesystem* (como ya dijimos, sistema de archivos). Por ejemplo, considere los links . y .. que aparecen en cada directorio. Para tener una acabada idea de qué directorio realmente es ..m primero vea el número de inodo de /usr/local

```
$ ls -id /usr/local
5120   /usr/local
```

Como ve, el directorio /usr/local tiene el número de inodo 5120. Ahora revise el número de inodo de /usr/local/bin/.. :

```
$ ls -id /usr/local/bin/..
5120   /usr/local/bin/..
```

Entendiendo inodos, segunda parte

Como puede ver, /usr/local/bin/.. y /usr/local tienen el mismo número de inodo. Veamos como explicar esta revelación. Hasta recién, considerábamos que /usr/local era *el* directorio realmente. Ahora, hemos visto que el inodo 5120 es *el* verdadero directorio, y además hemos hallado dos entradas distintas (en distintos directorios también, llamadas enlaces o *links*), que apuntan a ese inodo. Tanto /usr/local/ como /usr/local/bin/.. son enlaces al inodo

5120. A pesar de que el inodo 5120 existe en sólo un lugar del disco duro, múltiples enlaces pueden apuntar a él: el inodo 5120 es la verdadera entrada en el disco.

Entendiendo inodos, tercera parte

De hecho, usted puede ver el número total de veces que el inodo 5120 es referenciado, usando el comando ls -dl :

```
$ ls -dl /usr/local
drwxr-xr-x    8 root    root        240 Dec 22 20:54 /usr/local
```

Si tomamos la segunda columna de la izquierda, veremos que el directorio /usr/local (inodo 5120) es referenciado ocho veces. Aquí va una lista de algunas rutas que, en mi sistema, apuntan a ese inodo:

```
/usr/local
/usr/local/.
/usr/local/bin/..
/usr/local/games/..
/usr/local/lib/..
/usr/local/sbin/..
/usr/local/share/..
/usr/local/src/..
```

---

mkdir

Veamos ahora brevemente el comando mkdir que puede ser usado para crear directorios nuevos. En ejemplo siguiente crea tres directorios nuevos, tic, tac y toe, todos dentro de /tmp:

```
$ cd /tmp
$ mkdir tic tac toe
```

Por defecto, el comando mkdir no crea directorios padre para usted; la ruta completa hasta el ante-último elemento debe existir previamente. De esta manera, si quiere crear los directorio ma/ra/villa, tendrá que enviar tres comandos separados de mkdir:

```
$ mkdir ma/ra/villa
mkdir: no se puede crear el directorio "ma/ra/villa": No such file or directory
$ mkdir ma
$ mkdir ma/ra
$ mkdir ma/ra/villa
```

mkdir -p

Sin embargo, mkdir tiene una opción -p que le indica crear cualquier directorio padre faltante, como se ve a continuación:

```
$ mkdir -p asi/es/mas/facil
```

Bastante directo, ¿no?. Para aprender más sobre el comando mkdir escriba man mkdir, y lea la página de manual. Esto servirá para casi todos los comandos que veremos aquí (por ejemplo man ls), excepto para cd ya que este es *built-in* (comando interno) de bash.

touch

Ahora, vamos a revisar con rapidez los comandos cp y mv que sirven para copiar, renombrar y mover archivos y directorios. Para comenzar, primero veamos el comando touch (tocar) para crear un archivo en /tmp:

```
$ cd /tmp
$ touch copiame
```

Si el archivo ya existía, el comando touch actualiza el *"mtime"* del mismo (la sexta columna de la salida de ls -l). Si el archivo no existía, entonces se crea un archivo vacío nuevo. Tendrá entonces el archivo /tmp/copiame con tamaño cero.

echo

Ahora que el archivo existe, le agregaremos algunos datos. A esto podemos hacerlo mediante el comando echo (eco), que toma sus argumentos y los imprime en la salida estándar (*standard output*). Primero, sólo el comando echo:

```
$ echo "primerarchivo"
primerarchivo
```

echo y redirección

Ahora, el mismo comando echo pero redireccionando su salida:

```
$ echo "primerarchivo" > copiame
```

El signo mayor > le dice al shell que escriba la salida de echo a un archivo llamado copiame. Este archivo será creado si no existiese, y será sobreescrito si existía previamente. Escribiendo ls -l, podemos ver que ahora el archivo copiame tiene 12 bites de tamaño, ya que contiene la palabra primerarchivo y el carácter *newline* (salto de línea):

```
$ ls -l copiame
-rw-r--r--    1 root    root          10 Dec 28 14:23 /usr/local
```

---

cat y cp

Para ver los contenidos de un archivo en una terminal, use el comando cat:

```
$ cat copiame
primerarchivo
```

Bien. Ahora podemos hacer una invocación básica del comando cp para crear el archivo mecopiaron, siendo este una copia del original copiame

```
$ cp copiame mecopiaron
```

Si investigamos, veremos que son distintos archivos, pues sus números de inodos son diferentes

```
$ ls -i copiame mecopiaron
648284 copiame    650704 mecopiaron
```

---

mv

Usemos ahora el comando mv para renombrar "copiame" a "merenombraron". El número de inodo permanecerá igual; sin embargo, el nombre del archivo que apunta a él si cambiará:

```
$ mv copiame merenombraron
$ ls -i merenombraron
648284 merenombraron
```

Un número de inodo de un archivo movido o renombrado permanecerá igual mientras el archivo resida en el mismo *filesystem* que el archivo fuente. (Veremos más de cerca los sistemas de archivos en la Parte 3 de esta serie de tutoriales). Veamos otra manera de usar el comando mv, ya que este comando, además de permitirnos renombrar archivos, nos permite mover uno o más archivos hacia otra ubicación en la jerarquía de directorios. Por ejemplo, para mover /var/tmp/miarchivo.txt a /home/lucasperez, escribiré:

```
$ mv /var/tmp/miarchivo.txt /home/lucasperez
```

Después de tipear ese comando, miarchivo.txt será movido a /home/lucasperez/miarchivo.txt. Si /home/lucasperez está en un sistema de archivos distinto del de /home/lucasperez, el comando mv se encargará de copiar miarchivo.txt al nuevo sistema de archivos, y luego borrar el primero de su sistema de archivo. Como usted estará imaginando, cuando miarchivo.txt se traslada entre sistemas de archivos, el nuevo archivo miarchivo.txt en la nueva ubicación tendrá un nuevo número de inodo. Esto es porque cada sistema de archivos tiene su propio conjunto de inodos. También podemos usar el comando mv para mover varios archivos a un directorio. Por ejemplo, para mover miarchivo1.txt y miarticulo2.txt a /home/lucasperez, puede tipear:

```
$ mv /var/tmp/miarchivo.txt /var/tmp/miarticulo2.txt /home/lucasperez
```

4. Sección 4. Creando enlaces y removiendo archivos.

Enlaces permanentes (*Hard links*)

Hemos mencionado ya el término *link* (enlace) cuando nos referimos a la relación entre las entradas en los directorios (los nombre que tipeamos) y los inodos (el número de índice en el subyacente sistema de archivos que usualmente ignoramos). En realidad hay dos clases de enlaces en Linux, La clase de la que hemos discutido hasta aquí son llamados hard links(enlaces permanentes). Un número de inodo dado puede tener cualquier número de enlaces permanentes, y el inodo persistirá en el sistema de archivos hasta que el enlace permanente desaparezca. Cuando el último enlace permanente desaparece, y ningún programa mantiene el archivo abierto, Linux removerá el archivo automáticamente. Para crear un nuevo enlace permanente se utiliza el comando ln:

```
$ cd /tmp
$ touch primerenlace
$ ln primerenlace segundoenlace
$ ls -i primerenlace segundoenlace
15782 primerenlace   15782 segundoenlace
```

Continuando con enlaces permanentes

Como podrá ver, los enlaces permanentes trabajan en el nivel de los inodos para apuntar a un archivo en particular. En los sistemas Linux, los enlaces permanentes, tienen varias limitaciones. Uno es que sólo se pueden hacer enlaces permanentes a archivos, y no a directorios. Aunque . y .. son enlaces permanentes a directorios creados por el sistema, a usted ni siquiera como "root" se le permitirá crear otros. La segunda limitación es que no pueden expandirse a través de distintos sistemas de archivos. Esto significa que no puede crear un enlace permanente desde /usr/bin/bash hacia /bin/bash si sus directorios / y /usr pertenecen a distintos sistemas de archivos.

Enlaces simbólicos (*Soft links*)

En la práctica, los enlaces simbólicos (*symlinks* es como se suelen llamar) son utilizados más usualmente que los enlaces permanentes. Los *symlinks* son un tipo de archivo especial, donde el enlace se refiere a otro archivo a través de su nombre, en vez de hacerlo directamente al inodo. Los *symlinks* no previene

a un archivo de ser borrado, pues si elimina el archivo hacia el cual apunta el enlace, entonces el *symlink* deja de ser utilizable: se lo considera roto (*broken* en ingles)

Continuando con enlaces simbólicos

Un enlace simbólico puede ser creado agregando la opción -s al comando ln:

```
$ ln -s segundoenlace tercerenlace
$ ls -l primerenlace segundoenlace tercerenlace
-rw-rw-r--    2 juanperez   juanperez        0 Dec 31 19:16 primerenlace
-rw-rw-r--    2 juanperez   juanperez        0 Dec 31 19:16 segundoenlace
lrwxrwxrwx    1 juanperez   juanperez       13 Dec 31 19:17 tercerenlace ->
                                                           segundoenlace
```

En la salida de ls -l los enlaces simbólicos pueden ser diferenciados de los archivos comunes de tres formas. Por un lado, en la primer columna un carácter l indica que se trata de un enlace simbólico. Segundo, el tamaño de un enlace simbólico es el número de carácteres del archivo apuntado (segundoenlace en nuestro caso). Por último, en la última columna se ve el archivo apuntado (target más precisamente) precedido por una flechita (->) y el nombre del enlace.

Profundizando enlaces simbólicos, primera parte

Los enlaces simbólicos son por lo general mucho más flexibles que los permanentes. Usted puede crear un enlace simbólico que apunte a cualquier tipo de objeto del sistema de archivos, incluso directorios. Y como la implementación de los enlaces simbólicos está basada en rutas (y no inodos), es perfectamente posible crear un enlace simbólico que apunte a un objeto de otro sistema de archivos. Sin embargo, esto también puede hacer más difíciles de entenderlos.

Profundizando enlaces simbólicos, segunda parte

Considere una situación donde usted quiere crear un enlace en /tmp que apunte a /usr/local/bin. Deberia escribir algo como esto:

```
$ ln -s /usr/local/bin  bin1
$ ls -l bin1
lrwxrwxrwx    1 root    root        14 Jan 21 15:23 bin1 -> /usr/local/bin
```

O, alternativamente:

```
$ ln -s ../usr/local/bin bin2
$ ls -l bin2
lrwxrwxrwx    1 root    root        14 Jan 21 15:24 bin2 -> ../usr/local/bin
```

Profundizando enlaces simbólicos, tercera parte

Como podrá ver, los dos enlaces simbólicos recién creados apuntan al mismo directorio. Sin embargo, si el segundo enlace es trasladado a otro directorio, entonces quedará "roto" pues la ruta relativa será incorrecta:

```
$ ls -l bin2
lrwxrwxrwx    1 root    root        14 Jan 21 15:24 bin2 -> ../usr/local/bin
$ mkdir nuevodir
$ mv bin2 nuevodir
$ cd nuevodir
$ cd bin2
bash: cd: bin2: No such file or directory
```

Como el directorio /tmp/usr/local/bin no existe, entonces ya no podrá cambiar de directorio hacia bin2; en otras palabras, bin2 ahora está roto o *broken*

Profundizando enlaces simbólicos, cuarta parte

Es por esta razón que es entonces buena idea evitar crear enlaces simbólicos con rutas relativas. Sin embargo, hay muchos casos en que sí será útil. Considere por ejemplo el caso que usted quiera crear un nombre alternativo para un programa dentro de /usr/bin

```
$ ls -l /usr/bin/keychain
-rwxr-xr-x    1 root    root     10150 Dec 12 20:09 /usr/local/keychain
```

Profundizando enlaces simbólicos, quinta parte

Siendo el usuario root, usted podría querer crear un nombre alternativo para "keychain", como por ejemplo "kc". En este ejemplo, teniendo acceso de root (como queda evidenciado por el prompt de bash: un # en vez del clásico $). Necesitará poseer los privilegios de root porque a los usuarios comunes no les está permitido crear archivos en /usr/bin. Siendo root, puede crear un nombre alternativo para keychain de la siguiente manera:

```
# cd /usr/bin
# ln -s /usr/bin/keychain kc
# ls -l keychan
-rwxr-xr-x    1 root   root    10150 Dec 12 20:09 /usr/bin/keychain
# ls -l kc
-rwxr-xr-x    1 root   root       17 Oct 15 14:03 kc -> /usr/bin/keychain
```

En este simple ejmeplo, hemos creado un enlace simbólico llamado kc que apunta al archivo /usr/bin/keychain.

Profundizando enlaces simbólicos, sexta parte

Aunque la solución recién dada funcionará, puede llegar a traernos problemas si decidiéramos mover ambos archivos (/usr/bin/keychain y /usr/bin/kc) a /usr/local/bin:

```
# mv /usr/bin/keychain /usr/bin/kc /usr/local/bin
# ls -l /usr/local/bin/keychain
-rwxr-xr-x    1 root   root    10150 Dec 12 20:09 /usr/local/bin/keychain
# ls -l /usr/local/bin/kc
-rwxr-xr-x    1 root   root       17 Oct 15 14:03 kc -> /usr/bin/keychain
```

Como hemos usado un path absoluto para crear nuestro enlace permanente, kc aún sigue apuntando a /usr/bin/keychain, y como acababa de ser trasladado, /usr/bin/keychain no existe más, pues ahora su ruta es /usr/local/bin/keychain. Esto entonces significa que kc ahora es un *symlink* roto. Por lo visto tanto los *symlinks* con rutas absolutas como aquellos con rutas relativas tienen sus méritos, y usted deberá elegir el tipo de ruta apropiado para su aplicación. Frecuentemente ambos tipos de rutas funcionarán bien. El siguiente ejemplo funcionará aún después de que ambos archivos sean traslados:

```
# cd  /usr/bin
# ln -s keychain kc
# ls -l kc
lrwxrwxrwx    1 root   root        8 Oct 15 14:28 kc -> keychain
# mv keychain kc /usr/local/bin
-rwxr-xr-x    1 root   root    10150 Dec 12 20:09 /usr/local/bin/keychain
# ls -l /usr/local/bin/kc
lrwxrwxrwx    1 root   root        8 Oct 15 14:28 kc -> keychain
```

/usr/local/bin/kc apunta al programa keychain que está en el mismo directorio que kc. Ahora, escribiendo /usr/local/bin/kc usted puede ejecutar el

programa keychain.

---

rm

Ahora que conocemos cómo usar los comandos cp, mv y ln, es hora de aprender a remover objestos de nuestro sistema de archivos. Usualemente esto puede llevarse a cabo con el comando rm.

```
$ cd /tmp
$ touch arch1 arch2
$ ls -l arch1 arch2
-rw-r--r--    1 root   root        0 Oct 19 14:47 arch1
-rw-r--r--    1 root   root        0 Oct 19 14:47 arch2
$ rm arch1 arch2
$ ls -l arch1 arch2
ls: arch1: No such file or directory
ls: arch2: No such file or directory
```

Es importante destacar que en Linux, una vez que un archivo es removido (rm), es para siempre. Por esta razón, muchos administradores principiantes utilizan la opción -i cuando eliminan archivos. La opción -i le indica al comando rm que elimine archivos de modo interactivo (esto es, preguntando antes de eliminar cada archivo). Por ejemplo:

```
$ rm -i arch1 arch2
rm: remove regular empty file 'arch1'? y
rm: remove regular empty file 'arch2'? y
```

En el ejemplo, el comando rm pregunta por cada archivo especificado si se dedea realmente elinar el archivo. Si la respuesta es sí, deberá teclear una "y" (de *yes*) seguida de un Enter para cada pregunta. Si tipea una "n", el archivo entonces no será removido. Si usted hubiera hecho algo realmente mal, podrá abortar el comando en ejecución (rm -i en nuestro caso) tecleando Control-C. Al abortarlo, todos los cambios y modificaciones que hubiere ocasionado ya estarán hechos, pero impediremos que continue hasta el final. Si usted recién está comenzando a familiarizarce con el comando rm, puede ser de gran ayuda que agregue la siguiente linea a su archivo /.bashrc usando el editor de textos que prefiera, y luego salir y volver a entrar a la sesión ("desloguearse" y vover a "loguearse"):

```
alias rm="rm -i"
```

Ahora, cada vez que usted escriba rm, su shell bash lo convertirá automáticamente en el comando rm -i. De esta manera, rm siempre funcionará de modo interactivo.

---

rmdir

Para remover directorios, tenemos dos opciones. Una es eliminar primero todos los contenidos del directorio que queremos remover para luego usar el comando rm para borrar el directorio mismo:

```
$ mkdir midir
$ touch midir/arch1
$ rm midir/arch1
$ rmdir midir
```

Este método es comunmente llamado "eliminación de directorios para tontos". Todos los usuarios experimentados, como así también administradores ahorran trabajo utilizando el comando rm -rf que veremos a continuación.

---

rm y directorios

La mejor forma de remover un directorio es usando la opción de rm *recursive force* (recursivo y forzado). De esta manera se le indica al comando rm que remueva el directorio especificado como así también todos los objetos dentro del mismo:

```
$ rm -rf midir
```

Generalmente, rm -rf es el método preferido para elimiar un árbol de directorios. Tenga cuidado usando rm -rf ya que todo su poder pueder volverse en contra.