Manejo de archivos y directorios

Manejo de Archivos

El Shell de Linux proporciona muchos comandos para la manipulación de archivos. Esta sección lo guía a través de los comandos básicos del Shell necesarios para administrar archivos.

Creando archivos en Linux

A veces se puede encontrar con la necesidad de crear un archivo vacío. Por ejemplo, hay aplicaciones que esperan tener disponible un archivo de registro antes de iniciar a escribir sobre él. En estos casos, con el comando `touch` se puede crear un archivo con facilidad:

```
$ touch test_one
$ ls -l test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:17 test_one
$
```

El comando `touch` crea un archivo con el nombre especificado y asigna su nombre de usuario como propietario del archivo. Observe en el ejemplo anterior que el tamaño del archivo es cero bytes debido a que el comando `touch` crear un archivo vacío.

El comando `touch` también puede ser utilizado para cambiar el tiempo de modificación del archivo. Esto sin necesidad de cambiar su contenido:

```
$ ls -l test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:17 test_one
$ touch test_one
$ ls -l test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:35 test_one
$
```

La hora de modificación del archivo es cambiada de 14:17 a 14:35. Para cambiar solamente la hora de acceso, utilice el parámetro -a:

```
$ ls -l test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:35 test_one
$ touch -a test_one
$ ls -l test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:35 test_one
$ ls -l --time=atime test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:55 test_one
$
```

En el ejemplo anterior, observe que el comando `ls -l` no muestra la hora de acceso. Esto debido a que el comando muestra por defecto la hora de modificación. Para ver la hora de acceso al archivo, tiene que agregar un parámetro adicional, `--time=atime`. Después de agregar este parámetro, se muestra la modificación de la hora de acceso.

Crear archivos vacios y modificar las marcas de tiempo no es una tarea común en sistemas Linux. Sin embargo, el copiado de archivos es una tarea muy común, veamos a continuación.

Copiando archivos.

Copiar archivos y directorios de una ubicación a otra es una práctica común en la administración de sistemas. El comando `cp` proporciona esta característica.

En su forma mas básica, el comando `cp` utiliza dos parámetros – el objeto fuente y el objeto destino: `cp <source> <destination>`.

Cuando ambos parámetros son nombres de archivos, el comando `cp` copia el archivo origen en un nuevo archivo destino. El nuevo archivo actúa como un archivo independiente, con un nuevo tiempo de modificación:

```
$ cp test_one test_two
$ ls -l test_*
-rw-rw-r-- 1 christine christine 0 May 21 14:35 test_one
-rw-rw-r-- 1 christine christine 0 May 21 15:15 test_two
$
```

El nuevo archivo `test_two` muestra un tiempo de modificación diferente en relación al tiempo del archivo `test_one`. Si el archivo destino ya existe, es posible que el comando `cp` no lo indique. Es mejor utilizar la opción `-i` para forzar al Shell a que pregunte si usted desea sobre-escribir el archivo.

```
$ ls -l test_*
-rw-rw-r-- 1 christine christine 0 May 21 14:35 test_one
-rw-rw-r-- 1 christine christine 0 May 21 15:15 test_two
$
$ cp -i test_one test_two
cp: overwrite ‘test_two’? n
$
```

Si usted no responde, no se copia el archivo. También puede copiar un archivo en un directorio existente:

```
$ cp -i test_one /home/christine/Documents/
$
$ ls -l /home/christine/Documents
total 0
-rw-rw-r-- 1 christine christine 0 May 21 15:25 test_one
$
```

El nuevo archivo se encuentra ahora bajo el subdirectorio `Documents`, usando el mismo nombre del archivo original.

En el ejemplo anterior se utilizo una referencia de directorio absouluta, igualmente puede utilizarse una referencia de directorio relativa:

```
$ cp -i test_one Documents/
cp: overwrite ‘Documents/test_one’? y
$
$ ls -l Documents
total 0
-rw-rw-r-- 1 christine christine 0 May 21 15:28 test_one
$
```

En secciones anteriores leyó acerca de símbolos especiales que se utilizan en referencias a directorios relativas. Uno de ellos es el punto único (**.**), utilizado con el comando `cp`. Recuerde que el punto único representa su directorio de trabajo actual. Si usted necesita copiar un archivo cuyo nombre sea largo, el punto único ayuda a simplificar la tarea.

```
$ cp -i /etc/NetworkManager/NetworkManager.conf.
$
$ ls -l NetworkManager.conf
-rw-r--r-- 1 christine christine 76 May 21 15:55 NetworkManager.conf
$
```

Utilizar el punto único es mucho más fácil que escribir el nombre completo del archivo destino.

El parámetro `-R` es una opción muy potente del comando `cp`. Este permite copiar recursivamente el contenido completo de un directorio, todo en un solo comando.

```
$ ls -Fd *Scripts
Scripts/
$ ls -l Scripts/
total 25
-rwxrw-r-- 1 christine christine 929 Apr  2 08:23 file_mod.sh
-rwxrw-r-- 1 christine christine 254 Jan  2 14:18 SGID_search.sh
-rwxrw-r-- 1 christine christine 243 Jan  2 13:42 SUID_search.sh
$
$ cp -R Scripts/  Mod_Scripts
$ ls -Fd *Scripts
Mod_Scripts/  Scripts/
$ ls -l Mod_Scripts
total 25
-rwxrw-r-- 1 christine christine 929 May 21 16:16 file_mod.sh
-rwxrw-r-- 1 christine christine 254 May 21 16:16 SGID_search.sh
-rwxrw-r-- 1 christine christine 243 May 21 16:16 SUID_search.sh
$
```

Vea que el directorio `Mod_scripts` no existía antes de utilizar el comando `cp -R`. Este fue creado con el comando `cp -R`, junto con todo el contenido del directorio original. Observe también que todos los archivos en el nuevo directorio `Mod_scripts` tienen nuevas fechas de modificación. El nuevo directorio `Mod_scripts` es una copia completa del directorio `Scripts`.

También puede usar metacaracteres comodín en sus comandos `cp`:

```
$ cp *script Mod_Scripts/
$ ls -l Mod_Scripts
total 26
-rwxrw-r-- 1 christine christine 929 May 21 16:16 file_mod.sh
-rwxrw-r-- 1 christine christine 54  May 21 16:27 my_script
-rwxrw-r-- 1 christine christine 254 May 21 16:16 SGID_search.sh
-rwxrw-r-- 1 christine christine 243 May 21 16:16 SUID_search.sh
$
```

Este comando copia cualquier archivo terminado en `script` al directorio destino `Mod_scripts`. En este caso se copió solo un archivo: `my_script`.

Para el proceso de copia de archivos, además del punto único y de los metacaracteres comodín existe el tab-autocompletado. Veamos a continuación.

Usando el autocompletado con tab

Cuando se trabaja desde la línea de comandos, fácilmente se puede escribir de manera errónea un comando, archivo o directorio. De hecho, cuanto mas largo sea el nombre del archivo o directorio, mayor es la posibilidad de escribirlo mal.

Aquí es donde entra en juego el autocompletado con tab, este le permite comenzar a escribir el nombre de un archivo o directorio y luego de presionar la tecla tabulador, el Shell completa el nombre por usted.

```
$ ls really*
really_ridiculously_long_file_name
$
$ cp really_ridiculously_long_file_name  Mod_Scripts/
ls -l Mod_Scripts
total 26
-rwxrw-r-- 1 christine christine 929 May 21 16:16 file_mod.sh
-rwxrw-r-- 1 christine christine 54  May 21 16:27 my_script
-rw-rw-r-- 1 christine christine  0  May 21 17:08
really_ridiculously_long_file_name
-rwxrw-r-- 1 christine christine 254 May 21 16:16 SGID_search.sh
-rwxrw-r-- 1 christine christine 243 May 21 16:16 SUID_search.sh
$
```

En el ejemplo anterior, escribimos el comando `cp really` y presionamos la tecla tab, y el Shell completa automáticamente el resto del nombre del archivo. Por supuesto que debemos de escribir el nombre del directorio destino, pero aun así el autocompletado con tab nos ayuda a evitar posibles errores tipográficos.

El truco para usar el autocompletado con tab es darle al Shell suficientes caracteres del nombre del archivo o directorio para que pueda distinguirlos del resto de nombres contenidos en el directorio. Por ejemplo, si otro nombre de archivo comienza con `really`, al presionar la tecla `tab` no se completará automáticamente el nombre del archivo, en su lugar escuchará un pitido, de ser así puede presionar la tecla `tab` nuevamente y el Shell mostrará todos los nombres de archivos que comiencen con `really`.

Archivos de tipo enlace

El enlace de archivos es una característica disponible en sistemas Linux. Si necesita mantener dos o mas copias de un mismo archivo, en lugar de tener copias físicas separadas, puede utilizar una copia física y una o varias copias virtuales, llamados enlaces. Un enlace es un marcador de posición que apunta a la dirección real de un archivo. en Linux existen dos tipos de enlaces:

- Los enlaces simbólicos
- Los enlaces duros

Un enlace simbólico es un archivo físico que apunta a otro archivo ubicado en algún lugar de la estructura de directorio virtual. Los dos archivos unidos simbólicamente no comparten el mismo contenido.

Para crear un enlace simbólico, debe de existir un archivo origen. Para crear un enlace simbólico se utiliza el comando `ln` con la opción `-s`:

```
$ ls -l data_file
-rw-rw-r-- 1 christine christine 1092 May 21 17:27 data_file
$
$ ln -s data_file  sl_data_file
$
$ ls -l *data_file
-rw-rw-r-- 1 christine christine 1092 May 21 17:27 data_file
lrwxrwxrwx 1 christine christine    9 May 21 17:29 sl_data_file -> data_file
$
```

En el ejemplo anterior, observe que el enlace simbólico, `sl_data_file`, aparece como segundo argumento del comando `ln`. El símbolo → que aparece después del nombre del enlace simbólico muestra que está vinculado simbólicamente con un archivo de datos.

Tenga en cuenta también el tamaño del archivo del enlace simbólico en relación al archivo de datos. El enlace simbólico, `sl_data_file`, ocupa solamente 9 bytes, mientras que el archivo de datos ocupa 1092 bytes. Esto se debe que el archivo sl_data_file apunta al archivo data_file. No comparten contenido y físicamente son dos archivos distintos.

Otra manera de ver que estos archivos son físicamente distintos es viendo su número de `inodo`. El número de `inodo` de un archivo o directorio es un número de identificación único que el kernel de Linux asigna a cada objeto del sistema. Para ver el número de `inodo` de un archivo o directorio, utilice el comando `ls` con el parámetro `-i`:

```
$ ls -i *data_file
296890 data_file  296891 sl_data_file
$
```

En el ejemplo anterior se muestra que el número de `inodo` del archivo de datos es 296890, mientras que el número de `inodo` del archivo `sl_data_file` es 296891, por lo tanto, son archivos diferentes.

Un enlace duro crea un archivo virtual separado que contiene información y ubicación del archivo original, sin embargo, físicamente son el mismo archivo. Cuando referencia un archivo de enlace duro, es como si estuviera haciendo referencia al archivo original. Para crear un enlace duro, el archivo original debe existir, a diferencia que esta vez no se requiere que el comando `ln` vaya acompañado de ningún parámetro.

```
$ ls -l code_file
-rw-rw-r-- 1 christine christine 189 May 21 17:56 code_file
$
$ ln code_file hl_code_file
$
$ ls -li *code_file
296892 -rw-rw-r-- 2 christine christine 189 May 21 17:56
code_file
296892 -rw-rw-r-- 2 christine christine 189 May 21 17:56
hl_code_file
$
```

En el ejemplo anterior, utilizamos el comando `ls -li` para mostrar tanto los números de inodo como la lista larga de todos los archivos terminados en `*code_file`. Tomar en cuenta, que ambos archivos comparten el mismo número de inodo. Esto se debe a que físicamente son el mismo archivo. también notar que hay un contador de enlaces (la tercera columna de la lista). Además, el tamaño de los archivos es exactamente el mismo.

Los archivos enlazados le pueden parecer un concepto bastante confuso. Afortunadamente, la siguiente sección (renombrado de archivos) es mucho más fácil de entender.

Renombrando archivos

En el mundo de Linux, al cambio de nombre de archivos se le llama mover archivos. El comando `mv` mueve archivos y directorios hacia una nueva ubicación o a un nuevo nombre.

```
$ ls -li f?ll
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fall
296717 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fell
294561 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fill
296742 -rw-rw-r-- 1 christine christine 0 May 21 13:44 full
$
$ mv fall fzll
$
```

```
$ ls -li f?ll
296717 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fell
294561 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fill
296742 -rw-rw-r-- 1 christine christine 0 May 21 13:44 full
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fzll
$
```

Observe que al mover el archivo este cambia de nombre `fall` a nombre `fzll`, pero mantiene el mismo numero de inodo y de marca de tiempo. Esto es porque el comando `mv` solamente afecta el nombre del archivo.

También se puede utilizar el comando mv para cambiar la ubicación de un archivo:

```
$ ls -li /home/christine/fzll
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44
/home/christine/fzll
$
$ ls -li /home/christine/Pictures/
total 0
$ mv fzll  Pictures/
$
$ ls -li /home/christine/Pictures/
total 0
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fzll
$
$ ls -li /home/christine/fzll
ls: cannot access /home/christine/fzll: No such file or directory
$
```

En el ejemplo anterior, movimos el archivo `fzll` ubicado en el directorio `/home/christine` hacia el directorio `/home/christine/Pictures`. Nuevamente observar que no hubo cambios en el numero de inodo ni en la marca de tiempo del archivo.

El único cambio fue en la ubicación del archivo. vemos que dentro del directorio `/home/christine` ya no existe el archivo `fzll`, debido a que no se dejó ninguna copia en su ubicación original, lo que si hubiese ocurrido con el comando `cp`.

Utilizando el comando `mv` en un solo paso puede cambiar el nombre y la ubicación de un archivo:

```
$ ls -li Pictures/fzll
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44
Pictures/fzll
$
$ mv /home/christine/Pictures/fzll /home/christine/fall
$
$ ls -li /home/christine/fall
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44
/home/christine/fall
$
$ ls -li /home/christine/Pictures/fzll
ls: cannot access /home/christine/Pictures/fzll:
No such file or directory
```

En este ejemplo movimos el archivo `fzll` desde el subdirectorio, `Pictures`, hacia el directorio `/home/chrisitne`. Finalmente lo renombramos a fall, todo con una sola sentencia `mv`. Ni el valor de marcada de tiempo ni el número de inodo cambiaron. Solo se modificaron la ubicación y el nombre del archivo.

El comando mv se puede utilizar también para mover el directorio completo con sus contenidos.

```
$ ls -li Mod_Scripts
total 26
296886 -rwxrw-r-- 1 christine christine 929 May 21 16:16
file_mod.sh
296887 -rwxrw-r-- 1 christine christine  54 May 21 16:27
my_script
296885 -rwxrw-r-- 1 christine christine 254 May 21 16:16
SGID_search.sh
296884 -rwxrw-r-- 1 christine christine 243 May 21 16:16
SUID_search.sh
$
$ mv Mod_Scripts  Old_Scripts
$
$ ls -li Mod_Scripts
ls: cannot access Mod_Scripts: No such file or directory
$
```

```
$ ls -li Old_Scripts
total 26
296886 -rwxrw-r-- 1 christine christine 929 May 21 16:16
file_mod.sh
296887 -rwxrw-r-- 1 christine christine  54 May 21 16:27
my_script
296885 -rwxrw-r-- 1 christine christine 254 May 21 16:16
SGID_search.sh
296884 -rwxrw-r-- 1 christine christine 243 May 21 16:16
SUID_search.sh
$
```

Observe que el contenido del directorio no cambia. Lo único que cambia es el nombre del directorio. Después de ver como se renombran y mueven los archivos con el comando `mv`, nos damos cuenta d lo simple que es. La siguiente tarea es igualmente de fácil pero potencialmente es muy peligrosa.

Eliminando archivos

Es probable que en algún momento desee eliminar archivos existentes. Ya sea para limpiar el sistema de archivos o para eliminar paquetes de software, siempre se tiene la oportunidad de eliminar archivos. En el Shell bash el comando utilizado para eliminar un archivo es el `rm`. La forma mas simple del comando `rm` es:

```
$ rm -i fall
rm: remove regular empty file ‘fall’? y
$
$ ls -l fall
ls: cannot access fall: No such file or directory
$
```

Tenga en cuenta que el parámetro `-i` lo ayuda a estar seguro de querer eliminar el archivo. El Shell no cuenta con una papelera de reciclaje. Después de eliminar un archivo, desaparece para siempre. Por lo tanto, una buena práctica es agregar siempre el parámetro `-i` al usar el comando `rm`. Igualmente puede utilizar metacaracteres comodín para eliminar grupos de archivos. Sin embargo, por razones de seguridad, nuevamente utilice la opcion `-i`:

```
$ rm -i f?ll
rm: remove regular empty file‘fell’? y
rm: remove regular empty file‘fill’? y
rm: remove regular empty file‘full’? y
$
```

```
$ ls -l f?ll
ls: cannot access f?ll: No such file or directory
$
```

Para eliminar una gran cantidad de archivos sin tener que estar interviniendo para que sean eliminados, el comando `rm` utiliza el parámetro `-f` para forzar la eliminación, tener mucho cuidado al utilizarlo.

Manejo de directorios

Algunos comandos de Linux funcionan para archivos y para los directorios (comando `cp`) y algunos otros que funcionan solamente para los directorios. Para crear un directorio se requiere de un comando especifico, el cual se aborda en esta sección. También veremos la eliminación de directorios.

Creando un directorio

En Linux es fácil crear un nuevo directorio, solo utilice el comando `mkdir`:

```
$ mkdir New_Dir
$ ls -ld New_Dir
drwxrwxr-x 2 christine christine 4096 May 22 09:48 New_Dir
$
```

El sistema crea un nuevo directorio llamado `New_dir`. Observe que ahora el listado largo del directorio muestra que el registro del directorio comienza con la letra `d`. De esta manera indica que la entrada no es un archivo sino un directorio.

También puede crear archivos y directorios si es necesario. Sin embargo, si lo intenta con el comando `mkdir` recibirá el siguiente error:

```
$ mkdir New_Dir/Sub_Dir/Under_Dir
mkdir: cannot create directory ‘New_Dir/Sub_Dir/Under_Dir’:
No such file or directory
$
```

Para crear varios directorios y subdirectorios al mismo tiempo, solamente hay que agregar el parámetro -p:

```
$ mkdir -p New_Dir/Sub_Dir/Under_Dir
$
$ ls -R New_Dir
New_Dir:
Sub_Dir
```

```
New_Dir/Sub_Dir:
Under_Dir


New_Dir/Sub_Dir/Under_Dir:
$
```

La opción `-p` va creando los directorios padres a medida que se van necesitando. Un directorio padre es un directorio que contiene otros directorios en el siguiente nivel del árbol de directorios.

Después de saber como crear un directorio seguro se pregunta como eliminarlo. Esto es algo especialmente útil si creó el directorio en la ubicación incorrecta.

Eliminando directorios.

Borrar un directorio puede ser una tarea complicada por una razón. El Shell intenta protegernos de catástrofes accidentales tanto como sea posible. El comando básico para eliminar un directorio es `rmdir`:

```
$ touch New_Dir/my_file
$ ls -li New_Dir/
total 0
294561 -rw-rw-r-- 1 christine christine 0 May 22 09:52 my_file
$
$ rmdir New_Dir
rmdir: failed to remove ‘New_Dir’: Directory not empty
$
```

Por defecto, el comando `rmdir` solo puede eliminar directorios vacíos. Debido a que creamos un archivo, `my_file`, en el directorio `New_Dir`, el comando `rmdir` no puede eliminarlo.

Para solucionarlo, primero debemos eliminar el archivo. ahora con el directorio vacío podemos proceder a eliminar el directorio con el comando `rmdir`:

```
$ rm -i New_Dir/my_file
rm: remove regular empty file ‘New_Dir/my_file’? y
$
$ rmdir New_Dir
$
$ ls -ld New_Dir
ls: cannot access New_Dir: No such file or directory
```

El comando `rmdir` no cuenta con la opción `-i` para preguntar si está seguro de querer eliminar el directorio. Esta es una de las razones por las cuales `rmdir` puede eliminar solamente directorios vacíos.

Para eliminar directorios que no están vacíos se utiliza el comando `rm`. La opción `-r` hace que el comando descienda dentro del directorio, elimine el archivo y hasta entonces eliminar el mismo directorio.

```
$ ls -l My_Dir
total 0
-rw-rw-r-- 1 christine christine 0 May 22 10:02 another_file
$
$ rm -ri My_Dir
rm: descend into directory ‘My_Dir’? y
rm: remove regular empty file ‘My_Dir/another_file’? y
rm: remove directory ‘My_Dir’? y
$
$ ls -l My_Dir
ls: cannot access My_Dir: No such file or directory
$
```

Esto También funciona para descender bajo múltiples directorios y es especialmente útil cuando el directorio contiene muchos archivos y subdirectorios que necesitan ser eliminados.

```
$  ls -FR Small_Dir
Small_Dir:
a_file  b_file  c_file  Teeny_Dir/  Tiny_Dir/

Small_Dir/Teeny_Dir:
e_file

Small_Dir/Tiny_Dir:
d_file
$
$  rm -ir Small_Dir
rm: descend into directory ‘Small_Dir’?  y
rm: remove regular empty file ‘Small_Dir/a_file‘?  y
rm: descend into directory ‘Small_Dir/Tiny_Dir’?  y
rm: remove regular empty file ‘Small_Dir/Tiny_Dir/d_file’?  y
rm: remove directory ‘Small_Dir/Tiny_Dir’? y
rm: descend into directory ‘Small_Dir/Teeny_Dir’? y
rm: remove regular empty file ‘Small_Dir/Teeny_Dir/e_file’? y
rm: remove directory ‘Small_Dir/Teeny_Dir’? y
```

```
rm: remove regular empty file 'Small_Dir/c_file'? y
rm: remove regular empty file 'Small_Dir/b_file'? y
rm: remove directory 'Small_Dir'? y
$
$ ls -FR Small_Dir
ls: cannot access Small_Dir: No such file or directory
$
```

Aunque el método anterior es funcional, es algo incómodo. Tena en cuenta que se deben verificar cada uno de los archivos que se van a eliminar. En el caso de directorios que contienen muchos subdirectorios y archivos, esto puede volverse tedioso.

La solución para evitar tantos mensajes de precaución, en el caso de directorios con mucho contenido es utilizar el comando `rm` con los parámetros `-r` y `-f`:

```
$ tree Small_Dir
Small_Dir
├── a_file
├── b_file
├── c_file
├── Teeny_Dir
│   └── e_file
└── Tiny_Dir
    └── d_file

2 directories, 5 files
$
$ rm -rf Small_Dir
$
$ tree Small_Dir
Small_Dir [error opening dir]

0 directories, 0 files
$
```

El comando `rm -rf` no muestra ninguna advertencia, por lo que es una herramienta muy peligrosa, especialmente si cuenta con permisos de super usuario. Úsela con moderación y solo después de una triple verificación que le asegure que está haciendo exactamente lo que quiere hacer.

Note en el ejemplo anterior que hemos usado el comando `tree` para mostrar el árbol de directorios del directorio `Small_Dir`.

En las dos últimas sesiones hemos examinado la administración de archivos y directorios. Hasta ahora hemos cubierto todo lo que necesita saber sobre archivos, excepto ver su contenido.

Manejo de archivos

En Linux existen varias utilidades que permiten ver y buscar contenido dentro de un archivo sin necesidad de editarlo. Esta sección muestra algunos de los comandos disponibles para examinar archivos.

Ver el tipo de archivo

Antes de mostrar el contenido de un archivo, intentaremos saber que tipo de archivo es. Si intenta mostrar un archivo binario, seguramente observará en su emulador de terminal muchos caracteres ininteligibles.

El comando file permite identificar el tipo de archivo sobre el cual queremos realizar una tarea.

```
$ file my_file
my_file: ASCII text
$
```

En el ejemplo anterior el archivo es un archivo de tipo texto. El comando `file` no solo determina que el archivo contiene texto sino también el formato del código de caracteres de texto ASCII.

El siguiente ejemplo muestra un archivo que es simplemente un directorio. Por lo tanto esta es otra manera de como distinguir entre un archivo común y un directorio.

```
$ file New_Dir
New_Dir: directory
$
```

El siguiente ejemplo muestra un archivo de enlace simbólico. Notar que el comando `file` incluso muestra el archivo de datos al cual está vinculado.

```
$ file sl_data_file
sl_data_file: symbolic link to ‘data_file’
$
```

El siguiente ejemplo muestra la salida del comando file en el caso de archivos de tipo script. Aunque el archivo es de texto ASCII, usted puede ejecutarlo sobre el sistema.

```
$ file my_script
my_script: Bourne-Again shell script, ASCII text executable
$
```

Como ultimo ejemplo veremos el binario de un programa ejecutable. El comando file determina la plataforma para la que se compiló el programa asi como el tipo de bibliotecas que requiere. Esta es una característica especialmente útil si tiene un binario de una fuente desconocida.

```
$ file /bin/ls
/bin/ls: ELF 64-bit LSB  executable, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.24,
[...]
$
```

Ahora que conoce un método rápido de como ver el tipo de archivo, puede comenzar a ver el contenido de los archivos.

Viendo el contenido completo de un archivo

Si tiene un archivo de texto en su sistema Linux, probablemente quisiera ser capaz de ver el contenido dentro de este. Linux cuenta con tres comandos diferentes para hacer esta tarea:

1. Comando `cat`

   El comando es una herramienta útil para ver todo el contenido dentro de un archivo de texto.

```
$ cat test1
hello


This is a test file.



That we'll use to     test the cat command.
$
```

El comando anterior no muestra nada especial, solamente el contenido del archivo de texto. Sin embargo, el comando cat tiene algunos parámetros que le pueden parecer interesantes.

El parámetro -n muestra las líneas enumeradas.

```
$ cat -n test1
     1  hello
     2
     3  This is a test file.
     4
     5
     6  That we'll use to       test the cat command.
$
```

Esta es una característica muy útil cuando se trabaja con scripts. Si solo desea numerar las líneas que tienen texto utilice el parámetro -b:

```
$ cat -b test1
     1  hello

     2  This is a test file.



     3  That we'll use to       test the cat command.
$
```

Finalmente, sino desea que aparezcan caracteres de tabulación utilice el parámetro `-T`:

```
$ cat -T test1
hello

This is a test file.



That we'll use to^Itest the cat command.
$
```

El parámetro `-T` reemplaza cualquier tab en el texto con la combinación de caracteres `^I`.

Para archivos de texto muy grandes el comando `cat` puede ser algo molesto. El texto del archivo se desplaza rápidamente por la pantalla sin darle tiempo al usuario de poder leerlo. Afortunadamente, existe una forma simple para resolver este problema.

2. El comando `more`

   el principal inconveniente del comando `cat` es que no puede paginar. Para resolver este problema los programadores crearon el comando more. El comando more muestra un archivo de texto, pero se detiene después de mostrar cada pagina de datos. Escribamos el comando more `/etc/bash.bashrc` para producir la pantalla que se muestra en la figura 5.1

Figura 5.1 uso del comando more para mostrar archivos de texto

Observe en la parte inferior de la figura una etiqueta oscura que indica que todavía está dentro del comando more y que está al 56% de recorrido total del archivo. El comando more es una utilidad de paginación. Recuerde que en secciones anteriores hemos visto herramientas de paginación como el comando `man`. Al igual que utiliza el comando `man` para navegar por las paginas del manual, puede usar el comando `more` para navegar por un archivo de texto, al presionar la tecla espaciadora avanza pagina por pagina y con la tecla enter avanza línea por línea. Una vez finalice de navegar con el comando `more` puede salir presionando la tecla `q`.

A pesar de que el comando `more` ofrece mas funcionalidades que el comando `cat` sigue siendo un comando rudimentario, para funciones mas avanzadas tenemos el comando `less`.

3. El comando `less`

Por su nombre, no pareciera ser un comando mas avanzado que el comando `more`. Sin embargo, el nombre del comando `less` realmente es un juego de palabras y es una versión avanzada del comando `more` (su nombre viene de la frase “less is more”). El comando `less` proporciona funciones útiles para deslazarse hacia adelante y hacia atrás dentro de un archivo, así como capacidades avanzadas de búsqueda.

El comando `less` puede mostrar parte del contenido de un archivo antes de que este finalice de leer todo el archivo. Los comandos `cat` y `more` no soportan esta característica.

El comando `less` funciona de manera similar al comando `more`. mostrando una pagina de texto a la vez. Soporta los mismos comandos que el comando more, además de muchas más opciones.

Para ver las demás opciones del comando `less` vea su `man` page.

Ver el contenido de un archivo

Muy a menudo los datos que quiere ver dentro de un archivo se encuentran en la parte superior o en la parte inferior de este. Si la información está al inicio de un archivo muy grande, aun así, el `cat` tendría que esperar mucho tiempo para poder cargar el archivo completo. Si la información se encuentra en la parte inferior del archivo (tal es el caso de los archivos log), necesitaría paginar miles de líneas de texto solo para poder llegar a las ultimas entradas de texto. Afortunadamente, Linux tiene un par de comandos especializados para resolver estos problemas.

1. El comando `tail`

El comando `tail` muestra las últimas líneas de un archivo de texto. Por defecto, muestra las ultimas 10 líneas del archivo.

En el siguiente ejemplo tenemos un archivo de texto con 20 líneas. Primero utilizamos el comando `cat` y vemos que muestra el archivo completo.

```
$ cat log_file
line1
line2
line3
line4
line5
Hello World - line 6
line7
line8
line9
line10
line11
Hello again - line 12
line13
line14
line15
Sweet - line16
line17
line18
line19
Last line - line20
$
```

Ahora que ha visto el archivo completo. Puede ver el resultado de utilizar el comando `tail` para ver las ultimas 10 líneas del archivo:

```
$ tail log_file
line11
Hello again - line 12
line13
line14
line15
Sweet - line16
line17
line18
line19
Last line - line20
$
```

Si utiliza el parámetro `-n` puede cambiar el numero de líneas de salida. En el siguiente ejemplo se muestran únicamente las 2 ultimas líneas del archivo agregando el parámetro `-n 2`.

```
$ tail -n 2 log_file
line19
Last line - line20
$
```

El parámetro `-f` le permite echar un vistazo sobre un archivo, aunque esté siendo utilizado por otros procesos. El comando `tail` permanece activo y continúa mostrando nuevas líneas a medida que el proceso que lo tiene tomado continúa escribiendo sobre él. Esta es una forma común de monitorear en tiempo real los logs del sistema.

2. El comando `head`

El comando `head` exactamente lo que describe su nombre; muestra las primeras líneas de un archivo. por defecto muestra las primeras 10 líneas:

```
$ head log_file
line1
line2
line3
line4
line5
Hello World - line 6
line7
line8
line9
line10
$
```

Similar al comando `tail`, el comando `head` soporta el parámetro `-n` para cambiar el numero de líneas que desea mostrar.

```
$ head -5 log_file
line1
line2
line3
line4
line5
$
```

Por lo general el comienzo de un archivo nunca cambia, por lo que el comando head no soporta el parámetro `-f`.

Bibliografía:

Richard Blum, Christine Bresnahan. (January 2015). Linux Command Line and Shell Scripting Bible. Indianapolis, IN 46256, United States: Wiley.

Rob Kennedy. (Noviembre 2018). Essential Linux Command Line. United States: Kennedy Projects .