Preparación para el examen LPI 101

Tema 103

Comandos GNU & UNIX

Créditos y licencia de uso

Coordinación:
Manuel Guillán (xLekOx) lpi@xlekox.org
Oscar Casal (ocs) oscar@glug.es
Kiefer Von Jammo (Kiefer) kiefer@khrooon.net

Traducción:
Juan Maria Gil (Smooth) yo@juanmaria.com
Miguel Castiella (ruton) mcastiella@euskalnet.net
Carmen Eugenio (nemrac) meneiro@ono.com
Manuel Guillán (xLekOx) lpi@xlekox.org
Al H. Carril (Mandraker) spoofff@hotmail.com
Daniel Perelló (Baooroo) daniel_perello@hotmail.com
Dani Donisa (kasei) kasei@flashmail.com
Pablo Taboada (java) ptaboada@wanadoo.es

Maquetación y corrección:
Oscar Casal (ocs) oscar@glug.es
Rafael Díaz (fistipaldi) fistipaldi@teleline.es
Dani Donisa (kasei) kasei@flashmail.com
Manuel Guillán (xLekOx) lpi@xlekox.org
Kiefer Von Jammo (Kiefer) kiefer@khrooon.net
Kitano kitano@menta.net

Versión 1.0 (18-08-2004 18:00)

ÍNDICE

Índice de contenido

Tema 103.1

Trabajando en la linea de comandos

Introducción

En este primer tema se aprenderá el uso básico de la linea de comandos (shell). Usar comandos de ayuda, su formato, uso de diferentes parámetros, personalizar el entorno y usar el historial de comandos.

Los comandos que se verán en este tema son:

Archivos ocultos (empiezan por un .)
bash
echo
env
exec
export
man
pwd
set
unset
~/.bash_history
~/.profile

Así mismo se harán ejercicios sobre los mismos al final del tema, que serán muy parecidos a los realizados en los exámenes.

Este tema tiene un peso (importancia) de 5 de cara al examen final de la certificación LPI 101. El total de la suma de pesos de todos los temas es de 106.

Usando la Línea de Comandos

OBJETIVO: Trabajar de forma eficaz con la línea de comandos. Incluye la escritura de comandos válidos y las secuencias de comandos, usando substitución, y aplicando comandos recursivos a través de un árbol de directorios.

Una vez que te has logeado dentro del sistema Linux, te enfrentas con el shell. Éste aparece simplemente como una interface de línea de comando. A partir del prompt del shell escribimos comandos que son interpretados por el shell y enviados al sistema. El shell que estés corriendo en ese momento configura su prompt correspondiente. La mayoría de los sistemas Linux tienen como predeterminado el shell bash, y generalmente está configurado para mostrar el nombre de usuario, nombre del servidor y directorio actual de trabajo en el prompt. Un ejemplo de este prompt sería:

```
[angie@redhat /etc]$
```

En este ejemplo **angie** es el nombre de usuario, **redhat** es el nombre del servidor y **/etc** es el directorio actual de trabajo (llamado pwd – present working directory). El prompt para la shell de bash es el símbolo **$**. El prompt se configura a través del archivo /etc/bashrc. Por medio de este archivo se puede cambiar la configuración de lo que se muestra en el prompt. Cada usuario puede personalizar el prompt a su gusto, creando para ello el archivo de configuración personalizado en ~/.bashrc

Los comandos escritos en la línea de comandos no son enviados al sistema hasta que no se haya pulsado la tecla Enter. Esto permite editar los comandos antes de que se pasen al sistema. Los comandos escritos en la línea de comandos deben seguir una sintaxis específica. La primera palabra es el comando que se debe ejecutar; esta puede incluir una ruta absoluta, que es la ruta completa al comando, que debe terminar con el nombre del comando. A continuación deben figurar las opciones que son usadas por el comando. Los argumentos del comando van a continuación de las opciones. Cada uno de estos elementos se deben separar en la línea de comandos por medio de un espacio en blanco. El formato de un comando es el siguiente:

```
$ comando opciones argumentos
```

Recordatorio de examen: Es importante recordar esta sintaxis. Algunas preguntas del examen cuestionan sobre el correcto uso de comandos, opciones y argumentos.

Por ejemplo, tomando el comando ls. Este comando lista los archivos y directorios que hay en el pwd. Para ejecutar este comando, simplemente hay que teclear:

```
$ ls
```

Las opciones son códigos de una letra precedidos por un guión (-), y modifican la acción del comando. Se pueden combinar y usar varias opciones con un mismo comando. Las opciones son sensibles a mayúsculas y, en muchas ocasiones, una letra minúscula significará una opción diferente que su correspondiente mayúscula. Una opción muy importante disponible con muchos comandos es la -R, que especifica que el comando se debe ejecutar de forma recursiva a través del árbol de directorios.

Recordatorio de examen: La opción para la función recursiva de un comando es muy importante y aparecerá en el examen.

Si lo que quieres es listar los archivos y los directorios, y que los directorios aparezcan seguidos de una barra inclinada (/), se puede usar la opción -F (prueba el resultado).

```
$ ls -F
```

Ahora suponed que deseamos mostrar el contenido del directorio /etc desde otra localización. Además queremos que se muestre una barra inclinada tras todos los nombres de directorios que estén dentro de /etc. El directorio /etc se puede utilizar como argumento para el comando ls. Un argumento es otra opción que se puede utilizar con un comando. En el siguiente ejemplo, el argumento es un directorio que será examinado por el comando. Sería algo así:

```
$ ls -F /etc
```

Otro carácter especial que se puede utilizar cuando se introducen comandos es la barra invertida (\). Cuando se introduce a la derecha, antes de pulsar la tecla Enter, la barra invertida permite extender el comando a lo largo de varias líneas. La barra invertida provoca que el sistema ignore la pulsación de la tecla Enter, y trata todos los comandos como si estuvieran en una única línea. Esta función puede ser útil si tecleamos comandos muy largos, para poder partirlos en varias líneas. Un ejemplo sería el siguiente:

```
$ ls -F /etc \
ls -F /usr
```

En el mundo real: Para ejecutar comandos muy largos se utilizan scripts, en lugar de teclearlos directamente sobre la línea de comandos. Sin embargo, es importante recordar el uso de la barra invertida para el examen.

Otras opciones que se pueden usar con el comando ls pueden ser:

```
$ ls -l
```

Devuelve la lista de archivos y directorios en formato extenso, incluyendo nombre, privilegios de acceso, propietario, tamaño, fecha de última modificación, etc.

```
$ ls -a
```

Muestra todos los archivos y directorios del pwd, incluyendo los archivos ocultos.

```
$ ls *.gif
```

Muestra todos los archivos del pwd que terminen por ".gif".

```
$ ls ga*
```

Muestra todos los archivos y directorios del pwd que comiencen por "ga".

Completando Comandos

El shell bash incluye una características denominada “completado de comandos”. Esto nos permite teclear las primeras letras de un comando, pulsar la tecla tabulador, y dejar que el sistema complete el comando por nosotros. Si queremos ejecutar el comando dmesg para mostrar el buffer del kernel, podríamos teclear:

```
$ dm
```

y pulsar la tecla Tabulador, de forma que el sistema completará el comando:

```
$ dmesg
```

Si existe más de una coincidencia con la cadena tecleada antes de pulsar la tecla Tabulador, el sistema hará sonar un beep. Pulsando de nuevo la tecla Tabulador se mostrarán todas las posibilidades que coincidan con lo tecleado. Pulsar la tecla Esc dos veces produce el mismo efecto que pulsar la tecla Tabulador.

Conectando varios Comandos

En todos los ejemplos utilizados hasta ahora utilizamos la tecla Enter para informar al sistema de que el comando debía ser procesado. Sin embargo, no estamos limitados a ejecutar un único comando de cada vez.

Podemos ejecutar varios comandos, sin que estén conectados entre sí de ningún modo, tecleándolos en la misma línea y separándolos por punto y coma (;). Por ejemplo, es posible listar todos los archivos del directorio actual y la fecha de hoy tecleando:

```
$ ls ; date
```

El punto y coma es un carácter especial que siempre significa que hay varios comandos en la misma línea. Debido a que esto último tiene un sentido global, podemos prescindir de los espacios en blanco a ambos lados del punto y coma para obtener el mismo resultado (ls;date).

Si lo que hacen los comandos tiene algo en común – la salida de uno de ellos se convertirá en la entrada del siguiente – entonces podemos conectarlos usando una tubería (|). Por ejemplo, si la lista de de archivos de un directorio es muy larga como para poder verla en una sola pantalla, podemos ver una pantalla de cada vez usando:

```
$ ls -l | more
```

De este modo, la salida del comando ls -l será la entrada del comando more. Si falla la primera parte de la línea de comando, no se podrá ejecutar la segunda.

Comodines

Los comodines son caracteres que se utilizan en lugar de otros caracteres que el sistema rellena. Los dos comodines más frecuentes son el asterisco (*) y la interrogación (?). Aunque en ocasiones se confundan, su significado es diferente y producirán resultados totalmente distintos.

El asterisco significa ninguno, alguno o todos los caracteres:

```
$ ls s*
```

Este comando mostrará todas las entradas (archivos o directorios) dentro del directorio actual que comiencen con la letra s, y que tengan cualquier número de caracteres a continuación (incluyendo ninguno). Un posible resultado del comando puede ser:

```
s sa sam samp sampl sample samples samples.gif
```

Hay que prestar atención a que el comando encuentra la “s” sola y la “s” seguida de cualquier número de caracteres a continuación. En contraste, la interrogación (?) es un contenedor para un y sólo un carácter. Utilizando las mismas posibilidades que antes, el comando:

```
$ ls s?
```

Encontrará entradas (archivos y directorios) dentro del directorio actual que comiencen por la letra “s” y que únicamente tengan una letra más. El resultado sería:

```
sa
```

Si quisiésemos encontrar las entradas que comiencen por s y cuyo nombre tenga 5 caracteres en total, utilizaríamos:

```
$ ls s????
```

En resumen, el asterisco significa todos o ninguno, y el interrogante siempre significa uno. Estos dos comodines no son excluyentes, de modo que se pueden combinar según las necesidades. Por ejemplo, para encontrar sólo los archivos que tengan una extensión de tres letras dentro del directorio actual, utilizaremos:

```
$ ls *.???
```

Para complicar un poco más las cosas también podemos utilizar los corchetes ([]) para especificar posibles valores. Todos los valores posibles deben estar dentro de los corchetes, y el shell los tratará individualmente:

```
$ ls [de]*
```

Este ejemplo encontrará todas las entradas que comiencen por “d” o por “e” y que contengan un número ilimitado de caracteres. Para encontrar las entradas de longitud de 3 caracteres que comiencen por “d” o por “e”, utilizaremos:

```
$ ls [de]??
```

El número de caracteres que podemos incluir dentro de los corchetes es teóricamente ilimitado. Sin embargo, si lo que queremos es encontrar todas las entradas que comiencen por una letra minúscula pero no por un número u otro carácter, podemos utilizar [abcdefghijklmnopqrstuvwxyz]. Debido a que esto es un rango, una forma mucho más simple de obtener el mismo resultado es poniendo:

```
$ ls [a-z]*
```

Los rangos no tienen que ser series completas de números o caracteres, podemos expresar subconjuntos de ellos. Por ejemplo, si queremos buscar entradas que comiencen por alguna letra entre la "d" y la "t", podemos utilizar indistintamente [defghijklmnopqrst] o [d-t]. Si la entrada puede comenzar por esas letras tanto en mayúsculas como en minúsculas, podemos usar [DEFGHIJKLMNOPQRSTdefghijklmnopqrst] o [D-Td-t]

Otros ejemplos son:

- Todas las letras (mayúsculas y minúsculas): [A-z]
  [A-z] es lo mismo que [A-Z] y [a-z]
- Todos los números: [0-9]
- Cualquier carácter que no sea un número: [!0-9]
- Cualquier carácter que no sea una letra: [!A-z]

Alias de comandos

Aunque el sistema operativo y la shell ofrecen multitud de comandos y utilidades, puedes crear alias con nombres que tengan más sentido para ti o que sean más pequeños y así teclear menos caracteres. Por ejemplo, si estás familiarizado con la línea de comandos de Windows o del DOS estarás acostumbrado a escribir **dir** para obtener una lista de ficheros de un directorio, para obtener lo mismo en Linux se escribiría **ls -l**. Podrías crear un alias de tal forma que cada vez que escribieses **dir** se ejecutase **ls -l**, utilizando la siguiente expresión:

```
alias dir="ls -l"
```

La sintaxis será siempre el alias seguido del comando que habrá de ejecutarse cuando se teclee el alias separados por el signo igual (=). En raras ocasiones podrías hacerlo sin encerrar el comando entre comillas, pero como norma general siempre deberías incluirlas.

¡Ojo! Para que los alias no se pierdan al finalizar la sesión deben añadirse al archivo .bashrc que se encuentra en el directorio home del usuario.

Path y otras variables

Cuando tecleas un comando en el prompt la shell primero busca entre sus comandos internos y de no encontrar el comando se buscará una utilidad(comando) externa con ese mismo nombre. Esto se realiza buscando en los directorios incluidos en la variable **PATH** y en el mismo orden en el que han sido definidos en dicha variable hasta que se encuentra el primer archivo ejecutable cuyo

nombre coincide con el tecleado, en caso de no encontrarse ninguno después de buscar en todos los directorios indicados aparecerá el mensaje (“command not found”).

A continuación se remarcan algunas cosas importantes que deben conocerse respecto al path:

- Puedes consultar el valor actual de PATH con el siguiente comando:

```
echo $PATH
```

En el path no se incluye por defecto el directorio actual, por tanto, podrías tener un fichero ejecutable en tu directorio, ver que está ahí (tecleando **ls**) pero al escribir su nombre obtener un error de “command not found”.

Para solucionar esto podrías escribir el pathname completo del fichero a ejecutar, añadir este directorio a la variable **PATH**, mover el fichero a un directorio incluido en dicha variable, o añadir el símbolo del directorio actual (.) a la variable **PATH**.

Las distintas entradas en el path irán separadas por dos puntos (:).

El orden de búsqueda en la variable **PATH** debería comenzar por los directorios donde se encuentran los comandos más comunes (los directorios bin) y terminar por los directorios donde se encuentren los comandos del usuario, si existiesen.
Para añadir un directorio al path puedes redefinir la declaración completa o, simplemente, añadir el nuevo directorio con el comando:

```
PATH=$PATH:{nuevo directorio}
```

Por tanto, para añadir el directorio /home/fulanito al path, el comando sería:

```
PATH=$PATH:/home/fulanito
```

Si quieres añadir una entrada tal que el directorio donde estás trabajando en ese momento esté siempre incluido en el path de búsqueda, escribe:

```
PATH=$PATH:./
```

Por motivos de seguridad te recomendamos que no hagas ésto último, pero si no tienes más remedio que hacerlo, colócalo siempre al final de la declaración de **PATH** como se comentó anteriormente y no al principio.

Variables Comunes

Podemos ver el valor de cualquier variable existente con el siguiente comando:

```
echo ${nombre_de_variable}
```

Por tanto, el comando:

```
echo $MAIL
```

mostrará el directorio de correo, $HOME, el directorio home, y así sucesivamente. Para obtener una lista completa de todas las variables definidas en el entorno puedes utilizar cualquiera de estos dos comandos: **env** y **set**.
Aunque las salidas de los mismos puedan variar ligeramente (variables del entorno contra variables locales), en su mayor parte la salida de **env** es un subconjunto de la salida de **set**.

Algunas de las variables que podemos visualizar son:

- **HOME**—El directorio donde inicias la sesión y a donde llegas si tecleas **cd** sin ningún parámetro adicional.
- **LINES**—El número de líneas de pantalla que se visualizarán entre cada pausa (comando more).
- **LOGNAME**— El nombre de usuario con el que has conectado.
- **PWD**— El directorio de trabajo actual, el directorio donde te encuentras en este momento.
- **SHELL**— El intérprete de comandos que estás utilizando.
- **TERM**— El tipo de terminal o emulación seleccionado.
- **USER**— Suele ser igual que **LOGNAME**

Como regla general, las variables del sistema siempre aparecen en mayúsculas.

Puedes cambiar el valor de las variables según lo necesites o añadir tus propias variables para que sean utilizadas por otros programas o desde la línea de comando. Por ejemplo para crear una nueva variable llamada **HOY**, solo tendrías que escribir:

```
HOY=Viernes
```

Ahora puedes ver el valor de esa variable escribiendo:

```
echo $HOY
```

El resultado es **Viernes**. Si ahora empleas el comando:

```
set
```

la variable aparecerá ahí, sin embargo si usas el comando:

```
env
```

no aparecerá. La variable ha sido creada localmente y solo podrá ser referenciada localmente. Para que sea accesible por subrutinas o procesos hijos debes utilizar el comando **export** que hará que pase de ser local a ser una variable del entorno:

```
export HOY
```

Esto pondrá la variable en el entorno donde podrá ser encontrada tanto localmente como por las

subrutinas y procesos hijos, la variable y su valor serán accesibles durante toda la sesión y se perderán una vez desconectes.

Para que el valor sea permanente debes añadir la definición a un perfil, por ejemplo puedes cambiar el valor de **PATH** para todos los usuarios del sistema, conectándote como root y editando el archivo /etc/profile modificando la línea donde se define la variable **PATH.** Ésto podrías hacerlo utilizando el editor vi con el siguiente comando:

```
vi /etc/profile
```

Si quisieses hacer la modificación solo para un usuario en particular tendrías que editar el archivo /home/nombre_del_usuario/.bash_profile (el punto al inicio del nombre del fichero indica que este es un fichero oculto, tendrías que teclear ls -a para poder verlo).

Las modificaciones en los perfiles solo se harán efectivas tras la desconexión y nueva conexión del usuario.

Para cambiar el valor de una variable, simplemente, vuelve a definirla con el nuevo valor:

```
HOY=Lunes
```

Como ya la habíamos exportado anteriormente ya no tenemos por que hacerlo otra vez y el nuevo valor quedará accesible tanto localmente como en el entorno.

Si fuese necesario eliminar una variable puedes utilizar el comando **unset**.

Configurando el prompt

Entre las variables presentes y definibles, están aquellas que definen el prompt. El prompt es el mensaje que visualiza la shell cuando está lista para recibir un comando.

Los prompts por defecto incluyen:

- $ El último carácter para sh, bash, y ksh
- % El último carácter para csh y zsh
- > El último carácter para tcsh

El prompt primario puede ser tanto la variable **PS1** o la variable **prompt**, dependiendo de que shell estés utilizando. En bash, un valor típico para **PS1** sería:

```
[\u@\h \W]\$
```

Componente por componente, **PS1** sería igual a lo siguiente:

- El corchete izquierdo ([)
- El nombre del usuario actual (**\u**)

- La arroba (@)
- El nombre del host actual (**\h**)
- Un espacio
- El directorio de trabajo actual (**\W**)
- El corchete derecho (])
- El signo del dolar (**$**)

Un ejemplo de este prompt sería:

```
[leko@pentimVIII home]$
```

La barra invertida (\) indica la utilización de un valor especial. Algunos de estos valores se muestran en la tabla 1.1.

TABLA 1.1

Valor	***Resultado***
\d	Fecha actual
\h	Nombre del host hasta el primer punto
\n	Interlínea
\s	Shell
\t	Hora actual
\u	Nombre usuario
\W	Directorio actual
\w	Ruta completa de directorios
\!	Número del historial (se comentará mas adelante)
\#	Número del comando
\$	Prompt por defecto—$ para los usuarios normales y # para root
****	Barra invertida literal
ABC	Texto libre
\[	Secuencia de caracteres no imprimibles
\]	Fin de secuencia de caracteres no imprimibles
$date	Salida del comando date (o cualquier otro)

Para modificar el prompt de forma permanente para todos los usuarios hace falta editar el fichero /etc/bashrc (como root) y modificar su contenido. Teniendo en cuenta que si un usuario tiene en su directorio personal el fichero ~/.bashrc se sobreescribirá el contenido en /etc/bashrc.

Si te fijas en las variables del sistema verás que, además de **PS1,** puedes encontrar **PS2**.
Anteriormente se comentó que podía terminarse una línea de comandos con una barra invertida para indicarle a la shell que aun no se había terminado de introducir todo el comando, si observamos un

ejemplo podemos ver lo siguiente:

```
[leko@pentimVIII home]$ ls -l *.gif \
> *.fig \
> *.bmp
```

Observa como el prompt cambió de lo indicado en **PS1** a un signo de mayor (**>**). Si hubiese seguido siendo el mismo **PS1** no podrías saber si se trata del mismo comando o de uno nuevo, por eso se cambia del prompt primario a otro secundario definido, precisamente por la variable **PS2**. Su valor podrá ser cambiado de la misma forma que el de **PS1,** incluyendo los valores especiales de la Tabla 1.1. La mayoría de las shells admiten tres o cuatro niveles de prompts.

Otras Variables

A estas alturas ya te habrás dado cuenta de que el signo del dolar ($) se utiliza para obtener el valor de una variable; si tienes una variable llamada **EJEMPLO**, puedes ver su contenido examinando **$EJEMPLO**.
Hay otras tres variables que pueden resultar prácticas para determinar tu entorno.

La primera—$$—muestra el ID del proceso correspondiente a la shell que se ejecuta actualmente:

```
echo $$
```

La segunda—$?—muestra el resultado del último comando ejecutado. Este resultado puede ser correcto (**0**) o incorrecto (**1**). Por ejemplo, el comando **ls** admite la opción **-F** que diferenciará entre ficheros y directorios insertando una barra invertida detrás del nombre de los directorios, sin embargo este comando no incluye la opción **-z**.
Dada esta información, en el ejemplo siguiente podremos comprobar la utilización de la variable $?

```
[leko@pentimVIII home]$ ls -F
Desktop\ sample snapshot01.gif snapshot02.gif
[leko@pentimVIII home]$ echo $?
0
[leko@pentimVIII home]$ ls -z
ls: invalid option — z
[leko@pentimVIII home]$echo $?
1
```

La tercera variable—$!—mostrará el ID del último proceso hijo que se inició en el background . Si no se hubiese iniciado ningún proceso de background, entonces la variable no tendría valor.

Los procesos se describen con mayor detalle en otro capítulo pero para esta explicación es suficiente con saber que poniendo un ampersand (**&**) al final de un comando le indicamos a la shell que ejecute dicho comando en el background:

```
[leko@pentimVIII home]$ echo $!
[leko@pentimVIII home]$ ls -F &
[leko@pentimVIII home]$ echo $!
19321
```

[leko@pentimVIII home]$

Usando el historial de comandos

El archivo del historial contiene una lista de los comandos introducidos en la línea de comando. La variable **HISTSIZE** define el número de comandos que se almacenarán en dicho archivo durante la sesión actual, esta variable puede estar definida tanto en /etc/profile como en ~/.profile (~ equivale al directorio home del usuario) y su valor por defecto es de 1000 entradas. El comando history muestra todas las entradas del archivo del historial que se guarda en ~/.bash_history.

Puedes navegar por las entradas del histórico con las flechas del teclado. La flecha hacia arriba mostrará las entradas anteriores, mientras que la flecha hacia abajo avanzará hacia adelante en el historial. De esta forma podemos ahorrarnos volver a escribir comandos que ya habíamos escrito con anterioridad. Mientras navegamos por los comandos podemos editarlos antes de ejecutarlos de nuevo.

La variable **HISTCMD** proporciona el índice dentro del historial comando que se está ejecutando. La variable **HISTFILE** especifica el nombre del fichero que contendrá el historial (~/.bash_history por defecto). La variable **HISTFILESIZE** especifica el máximo número de líneas que contendrá el fichero especificado en **HISTFILE** y, aunque suele tener el mismo valor que **HISTSIZE**, podría ser diferente ya que ésta última se refiere solo al histórico de la sesión actual y no al tamaño total del archivo histórico**.**

fc

La utilidad fc proporciona otra opción para editar los comandos del fichero histórico antes de ejecutarlos. La utilidad fc abre el editor de textos por defecto con el comando especificado y ahí podemos editarlo y salvarlo antes de ejecutarlo disponiendo de toda la potencia de un editor de textos. Podemos llamar a fc utilizando como parámetro el número del comando que queremos editar o, también, con un rango de comandos para, de esta forma, editarlos y ejecutarlos en conjunto. También es posible especificar el editor de textos a utilizar. Una vez que se ha llamado a fc el fichero de comandos puede ser editado como cualquier otro fichero de texto y los comandos editados se ejecutarán al salir del editor. La opción -l se utiliza para mostrar una lista de los valores especificados a continuación, podéis escribir, por ejemplo, fc -l 50 60 y obtendréis la lista de los comandos del 50 al 60 en el historial.

Tabla 1-2. Operadores de expansión del historial de comandos

Operador	***Descripción***
!!	También conocido como bang-bang,[1] este operador hace referencia al comando más reciente del historial.
!n	Hace referencia al comando número n del historial. Puedes utilizar el comando history para conocer estos números.
!-n	Hace referencia al comando actual menos n en el historial.
! cadena	Hace referencia al comando más reciente que comience por cadena.
!? cade	Hace referencia al comando más reciente que contenga cadena.
^ cadena1^cadena2	Sustitución rápida. Repite el último comando reemplazando la primera aparición de cadena1 por cadena2.

[1] Es común llamar bang al signo de admiración en los sistemas Linux y Unix.

– **A** l trabajar con Linux habrá ocasiones en las que necesites una información más amplia sobre la utilización de algún comando, utilidad o configuración del sistema. Aunque este y otros libros pueden ser muy útiles, ningún libro puede tener la información totalmente actualizada. Afortunadamente existen magníficos recursos para cuando necesitemos más información. Algunas de estas fuentes de información las podremos encontrar en nuestro sistema local, mientras que otras estarán disponibles en Internet. Este capítulo te informará sobre algunos de los lugares más útiles para buscar información. Este conocimiento te hará ahorrar mucho tiempo cuando trabajes con sistemas Linux y es esencial para la preparación del examen.

Obteniendo Ayuda con las Páginas Man

• Uso y administración de la documentación del Sistema Local. El uso y administración de los recursos de man y del material de /usr/doc/. Incluye el encontrar las páginas man más relevantes, buscar por las secciones de man, encontrar comandos y las páginas man relativas a éstos, configurar el acceso a las fuentes de man y al sistema man, utilizar la documentación del sistema almacenada en /usr/doc/ y otros lugares relacionados y determinar que documentación mantener en /usr/doc/.

Las páginas del manual o páginas man de Linux son el mejor lugar para resolver cuestiones relativas a la sintaxis y opciones de los comandos y utilidades del sistema. Los documentos de las páginas man están almacenados en un formato comprimido. El comando man descomprime y formatea estas páginas para su correcta visualización. Se accede a estas páginas utilizando el comando man seguido del nombre del comando que se quiere consultar. Un ejemplo de la sintaxis te este comando es el siguiente:

```
# man ls
```

Este comando buscará todas las páginas del manual relativas al comando ls. Cuando abras las páginas del manual lo primero que se visualizará es un banner con el comando y la página man que está siendo consultada. También se muestra aquí el logo FSF de Free Software Foundation.

Todo esto aparecería de esta forma:

LS(1) FSF LS(1)

Esto irá seguido del nombre del comando y de su función.

NAME

ls - list directory contents
Objective

A continuación se muestra la sintaxis del comando:

SYNOPSIS
ls [OPTION]... [FILE]...

A esto le sigue una descripción del comando. Tras la descripción se muestran y explican las opciones del mismo.

DESCRIPTION
List information about the FILEs (the current directory by
default). Sort entries alphabetically if none of -cftuSUX nor
--sort.
-a, --all
do not hide entries starting with .
-A, --almost-all
do not list implied . and ..
-b, --escape
print octal escapes for nongraphic characters
--block-size=SIZE
use SIZE-byte blocks
-B, --ignore-backups
do not list implied entries ending with ~

La página man termina con información reltiva al autor de la página, bugs conocidos e información sobre como reportar nuevos bugs, copyright, e instrucciones para obtener más información sobre el comando.

AUTHOR
Written by Richard Stallman and David MacKenzie.
REPORTING BUGS
Report bugs to <bug-fileutils@gnu.org>.
COPYRIGHT
Copyright (c) 1999 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR
A PARTICULAR PURPOSE.
SEE ALSO
The full documentation for ls is maintained as a Texinfo manual.
If the info and ls programs are properly installed at your site,
the command
info ls
should give you access to the complete manual.

GNU fileutils 4.0p March 2000 1

Para desplazarse hacia abajo por las páginas man se utiliza la barra espaciadora, que avanzará una página por cada pulsación. La tecla Q provoca la salida del proceso de visualización de la página.

Si quieres buscar algún texto dentro de la página man puedes utilizar las expresiones regulares, a continuación se muestra un ejemplo de como buscar la palabra option.

```
/option
```

Encontrando las páginas man

Las páginas man se almacenan en el sistema. La variable MANPATH indica la ubicación de estos archivos. Por defecto las páginas man se guardan en los siguientes lugares.

- /usr/man/man1
- /usr/man/man2
- /usr/man/man3
- /usr/man/man4
- /usr/man/man5
- /usr/man/man6
- /usr/man/man7
- /usr/man/man8
- /usr/man/man9

El significado de los números se comentará en la siguiente sección del capítulo, "Buscando secciones de las páginas man"

Pregunta de Examen: Asegurate que sabes la ubicación por defecto de los ficheros fuentes de las páginas man. Esta pregunta es muy probable que salga en el examen.

El usuario puede especificar un MANPATH diferente. Esto permitiría utilizar un conjunto diferente de páginas man. Esto es práctico porque algunos comandos podrían almacenar sus páginas man en lugares distintos a los estándar. El comando man admite distintas opciones, una de ellas permite usar un path distinto al indicado en MANPATH. Las opciones del comando man se muestran en la tabla 1-3.

Tabla 1-3 Opciones utilizadas con man

Opción	***Uso***
-C fichero-de-configuración	Indica un fichero de configuración distinto a /etc/man.conf.
-M path	Indica en que directorios se buscarán las páginas man.
-P paginador	Indica el paginador o el programa utilizado para formatear y visualizar las páginas man. El paginador por defecto es el indicado en la variable de entorno PAGER Los paginadores more y less son los más frecuentemente utilizados.
-S Lista-de-secciones	Indica una lista de las secciones a buscar separadas por dos puntos (:)
-a	Indica que han de mostrarse todas las entradas coincidentes y no solo la primera.
-c	Indica que la página fuente ha de ser reformateada.
-d	Indica que debe mostrarse información de debug en lugar de las páginas man.
-f	Indica que el programa man debe comportarse como el programa whatis. (se explicará mas adelante).
-h	Muestra información sobre el comando man.
-k	Indica que el programa man debe comportarse como el programa apropos.(se explicará mas adelante).
-K	Busca una cadena especificada en las páginas man. Por cada entrada encontrada se le pregunta al usuario si desea verla.
-m	Indica un conjunto alternativo de páginas man basado en el sistema especificado.
-w	Indica que ha de visualizarse el path de las páginas man en lugar de las páginas.

A continuación se muestra un ejemplo del uso de la opción -a. Ésta hace que las páginas coincidentes sean mostradas en el orden en el que han sido encontradas. En primer lugar se le muestra al usuario la entrada correspondiente a crontab en la sección uno. Cuando el usuario pulsa la tecla Q para salir de ésta página , se mostrará la entrada encontrada en la sección cinco.

```
# man –a crontab
```

La opción -w es práctica para encontrar la ubicación de las entradas de las páginas man. Si utilizásemos esta opción con la utilidad crontab obtendríamos lo siguiente:

```
# man -w crontab
/usr/man/man1/crontab.1.gz
```

Pregunta de Examen: Asegurate que conoces las opciones de búsqueda y sus funciones, entre ellas -a, -K, y -k.

Buscando secciones de las páginas man

La información de las página man de Linux están contenidas en un conjunto de archivos. Estos archivos están agrupados en secciones y cada sección contiene un tipo específico de información. La Tabla 1-4 lista éstas secciones y su uso:

Tabla 1-4 Secciones de las Páginas man

Sección	***Uso***
1	Comandos y aplicaciones del usuario.
2	Llamadas del sistema y errores del Kernel.
3	Llamadas a librerías.
4	Drivers de dispositivos y protocolos de red.
5	Formatos estándar de archivos.
6	Juegos y demos.
7	Ficheros y documentos misceláneos.
8	Comandos de administración del sistema.
9	Especificaciones e interfaces oscuros del kernel.

Cuando se le pasa un argumento al comando man, éste busca dentro de las secciones siguiendo un orden específico y se retorna la primera coincidencia. El orden de búsqueda por defecto es el siguiente:
1, 8, 2, 3, 4, 5, 6, 7, 9.
También es posible indicar la sección en la que quieres buscar. Si quisieses buscar información sobre la utilidad crontab en la sección cinco utilizarías el siguiente comando:

```
# man 5 crontab
```

Si usas la opción -a tal como se muestra en la Tabla 7-1 podrás examinar todas las entradas coincidentes. Siguiendo el ejemplo de la utilidad crontab tendrías que utilizar el siguiente comando:

```
# man -a crontab
```

Buscando con whatis

La utilidad whatis no se menciona específicamente en los objetivos del examen, pero conocer su uso puede venir bien en el examen. La utilidad whatis se usa para buscar entradas coincidentes en la base de datos whatis. Esta base de datos se crea utilizando el comando /usr/bin/makewhatis. Esta base de datos contiene las descripciones cortas que se encuentran en las páginas man de los comandos del sistema. Un ejemplo de su uso es el siguiente:

```
# whatis passwd
  passwd (1) - update a user's authentication tokens(s)
  passwd (1ssl) - compute password hashes
  passwd (5) - password file
  passwd.nntp [passwd] (5) - passwords for connecting to remote NNTP servers
```

Como puedes ver en este ejemplo el comando passwd tiene entradas en las secciones uno y cinco

de las páginas man. También ha sido encontrado en la sección uno de las páginas man del comando ssl.

El comando man -f busca en esta base de datos entradas coincidentes con la palabra clave indicada. A continuación tenemos un ejemplo de la salida producida por este comando.

```
# man -f passwd
  passwd (1) - update a user's authentication  tokens(s)
  passwd (1ssl) - compute password hashes
  passwd (5) - password file
  passwd.nntp [passwd] (5) - passwords for connecting to remote NNTP servers
```

Estos comandos realizan la misma búsqueda. Se muestran los comandos y las páginas man donde han sido encontrados. Esto puede ser práctico para localizar secciones de páginas man y variantes de los comandos.

Buscando con apropos

Al igual que whatis, el comando apropos no se menciona específicamente en los objetivos del examen, pero conocer su uso puede venir bien en el examen. Y también como la utilidad whatis, el comando apropos utiliza la base de datos whatis. Este comando se emplea para buscar tanto los nombres de comando como las descripciones para la palabra clave indicada. A continuación vemos un ejemplo del comando apropos:

```
# apropos password
  chpasswd (8) - update password file in batch
  gpasswd (1) - administer the /etc/group file
  htpasswd (1) - Create and update user authentication files
  nwpasswd (1) - Change a user's password
  passwd (1) - update a user's authentication tokens(s)
  passwd (1ssl) - compute password hashes
  passwd (5) - password file
  passwd.nntp [passwd] (5) - passwords for connecting to remote NNTP servers
  pg_passwd (1) - Manipulate the flat password file
  pwupdate (8) - updates passwd and shadow NIS map
  rpc.yppasswdd [rpc] (8) - NIS password update daemon
  smbpasswd (5) - The Samba encrypted password file
  smbpasswd (8) - change a users SMB password
  ypchfn [yppasswd] (1) - change your password in the NIS database
  ypchsh [yppasswd] (1) - change your password in the NIS database
  yppasswd (1) - change your password in the NIS database
```

A continuación tenemos un ejemplo del comando man -k :

```
# man -k password
  chpasswd (8) - update password file in batch
  gpasswd (1) - administer the /etc/group file
  htpasswd (1) - Create and update user authentication files
```

nwpasswd (1) - Change a user's password
passwd (1) - update a user's authentication tokens(s)
passwd (1ssl) - compute password hashes
passwd (5) - password file
passwd.nntp [passwd] (5) - passwords for connecting to remote NNTP servers
pg_passwd (1) - Manipulate the flat password file
pwupdate (8) - updates passwd and shadow NIS map
rpc.yppasswdd [rpc] (8) - NIS password update daemon
smbpasswd (5) - The Samba encrypted password file
smbpasswd (8) - change a users SMB password
ypchfn [yppasswd] (1) - change your password in the NIS database
ypchsh [yppasswd] (1) - change your password in the NIS database
yppasswd (1) - change your password in the NIS database

Como puedes ver estos comandos hacen lo mismo. Esto puede ser práctico cuando busques comandos a partir de determinadas palabras clave.

Configurando el acceso a las páginas man

Como se mencionó con anterioridad en este capítulo, en el directorio /usr/man se guardan por defecto los ficheros fuente de las páginas man. La variable de entorno MANPATH puede ser utilizada para cambiar el path de búsqueda por defecto de los ficheros fuente de las páginas man. La variable MANPATH sobreescribirá el path de búsqueda por defecto de las páginas man, así que es importante incluir el path de las páginas man existentes. Abajo tenemos un ejemplo de una definición de la variable MANPATH añadida a un fichero /home/user/.profile.

```
Export MANPATH=/usr/local/man:/usr/man/preformat:/usr/man:/usr/X11R6/man
```

La mayoría de los documentos almacenados en /usr/man están comprimidos y sin formatear. El comando man utiliza el fichero /etc/man.config para obtener información acerca de la correcta visualización de esos fichero.
Este fichero contiene la definición de MANPATH, así como, información relativa a las opciones de compresión, formateo y paginación. Mediante la opción -C mostrada en la Tabla 7-1, podemos especificar un fichero de configuración diferente.
El comando man se encuentra en /usr/bin. Este directorio debe estar incluido en la variable de entorno PATH, de lo contrario deberemos ejecutar el comando utilizando el path absoluto /usr/bin/man.

Tema 103.2

Procesando cadenas de texto usando filtros

Introducción

Este capítulo cubre la mayoría de las herramientas de procesado de textos disponibles en sistemas Linux. Éstas incluyen diversas utilidades de filtrado, que se usan para buscar y cambiar archivos, así como las herramientas de entrada y salida. Es necesario comprender el uso de estas herramientas, debido a que son especialmente útiles en las labores administrativas diarias. Por ejemplo, usando la herramienta sed para buscar y reemplazar texto en scripts, nos permite hacer cambios de un modo mucho más sencillo de lo que sería si tuviésemos que hacerlos de forma manual. Por otro lado, aparecerán seguro entre las preguntas del examen, así que asegúrate de entender bien las diferencias y usos de cada herramienta.

Los comandos que se verán en este tema son:

- cat
- cut
- expand
- fmt
- head
- join
- nl
- od
- paste
- pr
- sed
- sort
- split
- tac
- tail
- tr
- unexpand
- uniq
- wc

Así mismo se harán ejercicios sobre los mismos al final del tema, que serán muy parecidos a los realizados en los exámenes.

Este tema tiene un peso (importancia) de 6 de cara al examen final de la certificación LPI 101. El total de la suma de pesos de todos los temas es de 106.

GNU y comandos Linux

Procesa cadenas de texto usando filtros de procesamiento de textos. Envía ficheros de texto y produce cadenas a través de utilidades de filtro de textos para modificar la salida de una manera más útil. Incluye el uso de comandos estándar de Unix que se encuentran en el paquete de utilidades de texto (textutils) GNU como sed, sort, cut, expand, fmt, head, join, nl, od, paste, pr, split, tac, tail, tr, y wc.

Linux ofrece una variedad de herramientas para usar, procesar y filtrar texto. Estas herramientas permiten buscar datos y manipularlos dependiendo de cual se use. Las herramientas pueden usarse desde la línea de comandos o localizadas en scripts, las cuales se ejecutan para realizar las tareas necesarias.

Cuando trabaje con ficheros de texto preferirá usar un paginador como more o less. Estas utilidades son usadas para mostrar el texto página a página. La utilidad cat puede usarse también para mostrar el contenido de un fichero de texto. Estas utilidades no son filtros pero a menudo son muy útiles cuando se trabaja con ellos.

Listando el contenido de un fichero

Un comando muy usado para listar los contenidos de ficheros de configuración o ficheros de pruebas de tamaño pequeño es el comando cat, un ejemplo sencillo de su uso sería listar un fichero de prueba llamado nombres.txt con la orden:

```
$ cat nombres.txt
```

saldría por pantalla el contenido del fichero.

Ordenando lineas de un fichero

La utilidad sort ordena las lineas de un texto y las muestra en la salida estándar. Se usa para ordenar, mezclar, y comparar líneas de ficheros o de la entrada estándar. Esta utilidad se usa con la siguiente sintaxis:

```
sort -opción nombre_fichero
```

Tabla 2-1 muestra las opciones más usadas con la utilidad sort.

Opción	***Uso***
-b	Ignora blancos
-c	Revisa si el fichero está ordenado
-d	Considera solo los caracteres alfanuméricos y ordena por directorio teléfono
-f	Ignora si los caracteres están en minúsculas o mayúsculas
-m	Une los ficheros ya ordenados sin reordenarlos
-M	Compara ficheros ordenados
-n	Ordena numéricamente
-o FILE	Escribe en un fichero específico de salida en lugar de la salida estándar
-r	Invierte los resultados
--help	Muestra la ayuda y salida
--version	Muestra versión y salida

Sigamos con un ejemplo del comando sort

Creamos un fichero de texto llamado list con las siguientes líneas:

```
3
eggs
5
Celestial Seasons
bread
32
Whiskas
butter
41
milk
ice
Centrum
90
flour
sugar
```

Ejecutamos el comando

```
$ sort list
```

y observamos el resultado, por defecto, se ordena la lista por orden numérico seguido de orden alfabético, con mayúsculas antes que las minúsculas. En el siguiente ejemplo, la opción -f hace que sort ignore mayúsculas:

```
$ sort -f list
```

En este otro ejemplo, dos ficheros se fusionan y se ordenan. El primer archivo sería por ejemplo el archivo prueba1 con el siguiente contenido:

angie
birdgrlmon
kim
monica
denise
lisa
desteve
jcfraggle
nikks
judy
netchickie

y el otro, prueba2 con este otro contenido:

katrina
crystalmoon
lma
loudhouse
trinityz
jill
nikki

```
$ sort prueba1 prueba2
```

Si esos dos archivos estuvieran ordenados se podría usar el comando:

```
$ sort -m prueba1 prueba2,
```

su salida sería una combinación de los 2 archivos sin reordenación.

También es posible ordenar ficheros por campos. Los campos se pueden separar por espacios o tabuladores y son numerados empezando por cero. Cuando se ordenan campos, el símbolo + precede al numero de campo con cada fichero separado por espacios.

El fichero nameslist contiene la siguiente lista:

Scott Bessler
Jason Nash
Angie Nash
Derek Stutsman
Jeff Arellano
Paul Ward
Alex Blauvelt
Peter Anapol
David Goolsby
Michael Craig
Johannes Erdfelt
Thomas McCanta
Joakim Erdfelt

Pete Gizzi
Neil Schroeder

En el siguiente ejemplo, el fichero nameslist se ordena por el segundo campo (+1) y entonces por el primer campo (+0):

```
$ sort +1 +0 nameslist
```

El uso de estos campos permite mucha flexibilidad en ordenar listas en ficheros. Es importante recordar que la utilidad sort no cambia el fichero original. La salida se envía a la salida estándar donde se puede visualizar o redireccionar hacia otro comando o fichero.

Cortando texto

La utilidad cut se usa para escribir partes seleccionadas de un fichero en la salida estándar. La utilidad cut también se puede usar para seleccionar columnas o campos desde ficheros específicos. Es posible seleccionar un trozo de una línea específica, varios trozos, o un rango. La tabla 2-2 cubre las diversas opciones usadas con la utilidad cut.

Tabla 2-2 : Opciones usadas con cut

Opción	***Uso***
-b	Escribe en la salida solo el rango de bytes especificados
-c	Escribe en la salida solo los caracteres especificados
-f	Escribe en la salida solo los campos especificados, delimitados por tabuladores
-help	Muestra ayuda y luego sale
-version	Muestra información de la versión y luego sale

A continuación un ejemplo del uso de estos rangos para escribir en la salida solo los primeros diez caracteres de cada línea:

```
$ cut -c 1-10 nameslist
```

La utilidad cut puede especificar que se muestren bytes, caracteres o campos de ficheros.

Pegando texto

La utilidad paste permite juntar texto desde múltiples ficheros. Las líneas correspondientes del fichero específico se escriben en la salida estándar con cada línea separada por un carácter tabulador. Las opciones usadas con la utilidad paste se muestran en la tabla 2-3.

Tabla 2-3 : Opciones usadas con paste

Opción	***Uso***
-s	Pega líneas desde un fichero a la vez
-d delimit-list	Usa los caracteres especificados en delimit-list consecutivamente en vez del carácter tab cuando separa ficheros mezclados.

El contenido del fichero nicks es el siguiente:

Banmage
Rexmortis
Jackyl
Dragonstr
Alchemist
Just_joe
Johan
Sting
Dave
Thomas
Netchick
Netjunkie
Pri
Zaphod
Lordram

A continuación un ejemplo del uso de la utilidad paste,

```
$ paste nameslist nicks
```

Nos daría como resultado los nombres seguidos de los nicks, suponiendo que estuvieran ordenados los archivos del mismo modo y cada línea perteneciese a una única persona, este sería el resultado esperado teniendo en cuenta los ficheros fuente, sin embargo la utilidad paste une linea a linea cualquier fichero, esté o no ordenado y se haga corresponder o no.

Convirtiendo tabuladores en espacios

Cuando se crea un fichero en GNU/Linux, se usa la tecla tab para justificar el texto, pero según la configuración del sistema/editor/entorno, el tamaño de esta justificación puede variar.
Esto puede cambiar el aspecto de un fichero se vea en un SO o en otro (o en un entorno o en otro). Para hacer que el fichero se vea igual sin importar la configuración de esta justificación, se pueden cambiar las 'tabulaciones' por un numero especifico de espacios. Esto causará que el fichero se vea igual en todos los sistemas.

La utilidad expand se usa para expandir los tabuladores en espacios. Esta utilidad puede trabajar con texto tanto de un fichero como de una entrada estándar. Las opciones para la utilidad expand se muestran en la tabla 2-4.

Tabla 2-4 : Opciones utilizadas con expand

Opción	***Uso***
-I	Convierte sólo los tabuladores de comienzo de la línea
-t	Utiliza una número para especificar los espacios a convertir
--help	Muestra la ayuda
--version	Muestra la información de la versión

La acción por defecto de la utilidad expand es convertir todos los tabuladores de un fichero en ocho espacios.

El contenido del fichero ejemplo3 es el siguiente: (Entre el nombre y la F o la M hay un tabulador de separación, como se ve a simple vista no queda con el aspecto de una tabla corriente).

```
Scott Bessler   F
Jason Nash      M
Angie Nash      M
Derek Stutsmann         M
Jeff Arellano   S
Paul Ward       M
Alex Blauvelt   S
Peter Anapol    M
David Goolsby           S
```

Al ejecutar el comando

```
$ expand ejemplo3
```

Alineamos el contenido del archivo de modo que las M's y las F's queden a la misma altura, si la opción por defecto no es suficiente podemos usar la opción -t con el número de espacios que queremos substituir el tabulador, con el fin de poner todas las filas a la misma altura:

```
$ expand -t 15 ejemplo3
```

Convirtiendo espacios en tabuladores

La utilidad unexpand realiza la función inversa que expand, de modo que los espacios se convierten en tabuladores, las opciones más comunes se pueden consultar en su manual:

```
$ man unexpand.
```

Formateando párrafos.

La utilidad fmt formatea cada párrafo en un fichero y envía la salida al output estándar. Este comando es utilizado para especificar la anchura de las líneas y junta o separa líneas en un esfuerzo para que estas tengan la misma longitud. El comando fmt intenta separar líneas al final de cada sentencia. Cuando esto no es posible, intenta romper la línea después de la primera palabra o antes de la última palabra de la sentencia. Las sentencias finalizan con una marca de periodo, exclamación o cuestión (.!?) seguidas de dos espacios o un retorno de carro (line feed). Se lee todo el párrafo antes de que se introduzca cualquier salto de línea.

La anchura por defecto que utiliza fmt para una línea es de 75 caracteres. El ancho por defecto puede ser modificado usando la opción adecuada en el comando fmt. La tabla 2-5 recoge algunos de los parámetros del comando.

Tabla 2-5: Opciones utilizadas con fmt

Opción	***Uso***
-c	Mantiene igual los espacios de principio de párrafo y alinea el párrafo con margen izquierda en la segunda línea
-t	Trabaja como -c excepto que los espacios de comienzo de la segunda línea sean igual que la primera, considerando la segunda línea como un párrafo de una línea
-s	Especifica que las líneas van a ser divididas y no juntadas
-u	Especifica que espacio uniforme se va a utilizar; esto reduce la separación entre todas las palabras a un espacio y a dos espacios entre las sentencias
-NUMERO o -w NUMERO	Establece la largura de la línea al NUMERO indicado
-p PREFIX	Especifica que las líneas que empiecen por PREFIX serán modificadas

El comando fmt deja igual las lineas en blanco, los espacios de principio de párrafo y el spacing, así cualquier formateo especial dentro del documento no es modificado. Abajo mostramos un ejemplo del comando fmt. Como puedes ver en el ejemplo, todas las líneas entre párrafos son mantenidas, y el comando hace que todas las líneas sean de 40.

El fichero de ejemplo (ejemplo4) contiene una única línea con el siguiente texto:

Linux offers a variety of tools to use for pricessing and filtering text. These tools enable you to search for data...

Con el comando

```
$ fmt -40 ejemplo4
```

se formatea dicho texto.

Borrando o sustituyendo caracteres

Hay veces en las que se quiere buscar en un documento caracteres específicos y luego borrarlos o reemplazarlos por otros. Un ejemplo sería un documento que utiliza mayúsculas y minúsculas, pero se prefiere que todo el documento esté en minúsculas. El comando tr hace esto utilizando la siguiente síntesis:

```
tr option set1 set2
```

Las opciones que se utilizan con tr están en la tabla 2-6

Tabla 2-6 : Opciones utilizadas con tr

Opción	***Uso***
-d	Borra un carácter especificado.
-s	Remplaza una secuencia de caracteres por un carácter.
--help	Muestra la ayuda
--version	Muestra la versión

Esto es un ejemplo del comando tr. Primero, el contenido del fichero es mostrado utilizando el comando more. El documento tiene una mezcla de mayúsculas y minúsculas. Todos los caracteres en mayúsculas son sustituidos por minúsculas.

```
$ more ejemplo5
mkdir Myfiles
cd Myfiles
ls -al MyFiles > AllofMyFiles
```

Con el comando

```
$ tr 'A-Z' 'a-z' < ejemplo5
```

tendríamos la salida que buscamos.

Viendo el comienzo de un fichero

El comando head nos permite ver el comienzo de un fichero. Por defecto nos muestra las 10 primeras líneas. Las opciones del comando head están en la tabla 2-7

Tabla 2-7 : Opciones utilizadas con head

Opción	***Uso***
-c NUMERO	Especifica el número de bytes a ser mostrados.
-n NUMERO	Muestra el NUMERO de líneas especificado.
-q	No muestra las cabeceras.
-v	Muestra las cabeceras.
--help	Muestra la ayuda
--version	Muestra la versión.

Esto es un ejemplo del comando head. La opción -v es usada para mostrar el nombre de fichero localizado en la cabecera.

```
$ head -n 3 -v namelist
```

Viendo el final de un fichero

Hay un comando que nos permite ver el final de un fichero. Como head, el comando tail muestra las últimas 10 lineas de un fichero por defecto. Hay además muchas opciones en el comando tail que se describen en la tabla 2-8

Tabla 2-8 : Opciones utilizadas con tr

Opción	***Uso***
-NUMERO	Especifica el número de líneas a ser mostrado.
+NUMERO	Especifica el número de líneas desde el comienzo a partir de donde empieza a mostrar.
--retry	Indica a las instrucción que se mantenga intentando abrir un fichero cuando este esté inaccesible.
-c NUMERO	Especifica el número de bytes a ser mostrados.
-f	Muestra las líneas y va mostrando líneas según se escriben en el fichero. Este comando puede venir bien para ficheros que crecen como por ejemplo los ficheros LOG.
-n NUMERO	Muestra el NUMERO de líneas especificado.
-q	No muestra las cabeceras.
-v	Muestra las cabeceras.
--help	Muestra la ayuda
--version	Muestra la versión.

Esto es un ejemplo del comando tail. Vamos a mostrar los 50 últimos bytes del fichero namelist.

```
$ tail -c 50 namelist
```

Juntando múltiples ficheros

El comando join en realidad busca en dos ficheros entradas comunes. Las entradas encontradas en los dos ficheros son mostradas en la salida estandar donde pueden ser redireccionadas a un fichero. Puedes combinar ficheros usando campos. El comando join usa campos de unión para combinar líneas de múltiples ficheros. Antes de usar el comando join, el fichero debe de comenzar con los campos de unión. Esto se consigue muchas veces con el comando sort basado en los campos que van a ser juntados. Así, si tu estás utilizando dos ficheros que contienen el nombre y el primer apellido y quieres juntar los ficheros utilizando el apellido, entonces los dos ficheros deben ser ordenados previamente utilizando el campo apellido.

El correcto funcionamiento del comando join es el siguiente:

```
$ join -options FILE1 FILE2
```

Algunas opciones del comando join:

Tabla 2-9 : Opciones utilizadas con join

Opción	***Uso***
-I	Especifica que caso es ignorado cuando se combinan los ficheros.
-1 FIELD	Especifica el campo en el fichero 1
-2 FIELD	Especifica el campo del fichero 2
-t char	Especifica el carácter separador del fichero de entrada y de salida.
-v FILE#	Se imprime una línea por cada línea no "pareada" encontrada en el fichero FILE#
--help	Muestra la ayuda
--version	Muestra la versión.

El siguiente es un ejemplo del comando:

```
$ join prueba1 prueba2
```

Dividiendo ficheros en varias partes

La utilidad SPLIT se usa para dividir un fichero largo en varios ficheros distintos. Esta utilidad crea ficheros de una cierta longitud, cuyo valor por defecto es de 1000 líneas, y los nombra de forma secuencial. Los nombres de los archivos están formados por un prefijo, de valor "x" por defecto, seguido por una combinación de letras que sigue el patrón de "aa", "ab", "ac", etc. Si se deben crear más de 676 ficheros, la sintaxis será "zaa", "zab", etc. Cuando no se especifica ningún fichero de entrada para la utilidad SPLIT, la entrada de datos estándar se utilizará por defecto.

La sintaxis correcta para la utilidad SPLIT es la siguiente:

```
$ split opciones FICHERO  PREFIJO
```

Hay varias opciones, mostradas en la tabla 2-10, que pueden ser utilizadas para personalizar la salida de la utilidad SPLIT.

Tabla 2-10 : Opciones utilizadas con SPLIT

Opción	***Uso***
-l LÍNEAS	Especifica el número de líneas de cada uno de los ficheros de salida.
-b BYTES	Especifica el número de bytes de cada uno de los ficheros de salida. Si el número está seguido por "b", la cantidad indicada será multiplicada por 512; si se utiliza una "k", la cantidad será multiplicada por 1024; y si se utiliza una "m", la cantidad será multiplicada por 1048576.
-C BYTES	Funciona de manera similar a la opción "-b". Mediante esta opción, se irán añadiendo líneas completas a cada uno de los ficheros de salida, sin superar el número de bytes indicado.
--verbose	Escribe un diagnóstico en el error estándar antes de que se abra cada uno de los ficheros.
--help	Muestra información de ayuda.
--version	Muestra información sobre la versión.

En el ejemplo siguiente, el fichero nameslist se divide en varios archivos cada uno de los cuales contiene cinco líneas.

```
$ split -l5 nameslist names
```

Listamos el contenido del directorio para ver los archivos nuevos

```
$ ls names*
```

Eliminando líneas repetidas de un fichero

Para eliminar las líneas repetidas y consecutivas de un fichero usamos el comando uniq, de modo que si tenemos en un fichero "a" las líneas:

```
1
3
4
4
3
```

Al ejecutar el comando

```
$uniq a
```

El resultado será:

```
1
3
4
3
```

ya que la única línea que se repite es la 3ª y la 4ª que tienen el número 4.

Otras opciones de este comando se ven en la tabla 2-11:

Tabla 2-11

Opción	***Función***
-c	Enumera el número de ocurrencias (líneas que se repiten)
-d	Solo conserva las líneas que se repiten
-u	Solo conserva las líneas que son únicas

Mostrando ficheros en otros formatos

Hay ocasiones en las que puede ser necesario mostrar ficheros que estén en formato NO-texto y para ello, puede usarse la utilidad OD. Cada una de las líneas de salida consiste en el OFFSET, que es el número de bytes desde el inicio del fichero, y los de grupos de datos del fichero de entrada que se muestran a continuación. Por defecto, OD escribe el OFFSET en octal, y cada uno de los grupos de datos del fichero son dos bytes que se escriben como un único número octal. Las opciones para esta utilidad se muestran en la tabla 2-12.

Tabla 2-12 Opciones utilizadas con OD

Opción	***Uso***
-A RADIX	Especifica la base utilizada para mostrar el offset. El valor del parámetro puede ser cualquiera de los siguientes: d se utiliza para decimal o se utiliza para octal x se utiliza para hexadecimal n se utiliza para especificar ninguna
-j BYTES	Especifica el número de bytes de entrada que deben ser descartados antes de empezar a mostrar datos.
-N BYTES	Especifica el número máximo de bytes a mostrar
-s [N]	Muestra cadenas de al menos N caracteres.
-t TIPO	Selecciona el formato en el que se mostrará la información de salida. TIPO es una cadena de caracteres formada por uno o más de los siguientes indicadores de caracteres: a para caracteres c para caracteres ASCII o secuencia de escape d para decimal con signo f para coma flotante o para octal u para decimal sin signo x para hexadecimal
-w BYTES	Estructura la salida con el número de bytes por línea indicado. El valor por defecto es 32 bytes.
--help	Muestra información y termina.
--version	Muestra información sobre la versión y termina.

A continuación se muestra un ejemplo del uso de la utilidad OD. En este ejemplo, el fichero

"prueba1" se muestra con el offset en formato hexadecimal y los datos en octal, con una amplitud de ocho bytes.

```
$ od -A x -w8 prueba1
```

Consejo para el examen: Se debe prestar atención a las utilidades poco utilizadas como OD o JOIN, ya que son un excelente material para el examen.

Convirtiendo ficheros para imprimir

La utilidad PR formatea y prepara ficheros para imprimir, escribiéndolos en la salida estándar, paginándolos y opcionalmente escribiéndolos en un formato de multicolumna. Adicionalmente, también puede unir ficheros, imprimiéndolos en paralelo, uno por columna. La sintaxis para esta utilidad es la siguiente:

```
pr -opciones FICHERO
```

Por defecto, en cada página se escribe un encabezado de cinco líneas: dos líneas en blanco, una línea con la fecha, el nombre del archivo y el contador de página, y dos líneas más en blanco. Igualmente, también se escribe un pie de página de cinco líneas. Hay numerosas opciones para especificar el formato producido con la utilidad PR, algunas de las cuales se muestran en la tabla 2-13.

Tabla 2-13 : Opciones utilizadas con PR

Opción	***Uso***
-COLUMNAS	Produce tantas columnas como el número COLUMNAS y equilibra el número de líneas en cada columna dentro de cada página.
-a	Imprime las columnas en horizontal en lugar de en vertical.
-d	Inserta un doble espacio en la salida
-f	Utiliza saltos de página en lugar de caracteres de nueva línea para separar páginas.
-h TEXTO	Utiliza el texto especificado en TEXTO en lugar del nombre del fichero dentro del encabezado.
-l LÍNEAS	Establece el número de líneas por página
-m	Imprime todos los ficheros en paralelo, uno por columna.
-N NÚMERO	Empieza contando por NÚMERO en la primera línea de la primera página impresa.
-w CARACTERES	Establece el ancho de página a un número de caracteres igual a CARACTERES (el valor por defecto es 72). Solamente se utiliza para formatos de salida multicolumna.
-W CARACTERES	Establece el ancho de página a un número de caracteres igual a CARACTERES siempre. El valor por defecto es 72.

Mostrando ficheros al revés

La utilidad TAC se utiliza para mostrar líneas de un fichero al revés donde, por defecto, una nueva línea empieza después del carácter de salto de línea. Éste es el opuesto al comando CAT, tanto en el nombre como en su funcionamiento.

La sintaxis para la utilidad TAC es la siguiente:

```
$ tac – opciones FICHERO
```

Las opciones disponibles se muestran en la tabla 2-14.

Tabla 2-14 : Opciones utilizadas con TAC

Opción	***Uso***
-b	Adjunta el separador al principio de la línea que le precede en el fichero, en lugar de hacerlo al final.
-r	Trata la cadena de caracteres del separador como una expresión regular.
-s SEPARADOR	Utiliza el carácter SEPARADOR en lugar del carácter de salto de línea como el separador de registros.

Mostrando estadísticas de un fichero

La utilidad WC cuenta el número de bytes, palabras separadas por espacios en blanco y saltos de línea para cada uno de los ficheros indicados. Se muestra una línea de resultados para cada uno de los ficheros, y si el fichero fue indicado como un argumento, muestra su nombre a continuación. Si se indica más de un fichero, la utilidad muestra una línea final indicando los resultados acumulativos con el texto “total”.

El orden en el que se muestran los resultados es el siguiente: en primer lugar los saltos de línea, luego las palabras y finalmente los bytes. Por defecto, cada resultado se muestra justificado a la derecha en un campo de siete bytes con un espacio en blanco entre cada uno de los resultados, de manera que los números y los nombres de los ficheros se alinean correctamente en columnas. Algunas de las opciones disponibles con WC se describen en la tabla 2-15.

Tabla 2-15 : Opciones utilizadas con WC

Opción	***Uso***
-c	Muestra únicamente el número de bytes
-w	Muestra únicamente el número de palabras
-l	Muestra únicamente el número de líneas
-L	Muestra la longitud de la línea más larga
--help	Muestra información de ayuda y termina
--version	Muestra información sobre la versión y termina

A continuación se muestran algunos ejemplos:

```
$ wc prueba1
```

```
$ wc -c prueba1
```

Añadiendo números de línea a un fichero

La utilidad nl es útil para mostrar los números de línea de un fichero. Se organiza el fichero de entrada en páginas lógicas y por defecto, el número de línea se inicializa a 1 al principio de cada una de ellas. Se tratan todos los ficheros de entrada como un único documento y no se inicializan los números de línea ni las páginas lógicas entre ficheros.

Una página lógica consiste en tres secciones separadas por una línea en blanco: encabezado, cuerpo y pie de página. Cualquier de ellas puede estar vacía y puede estar numerada de una forma distinta a las otras dos. El texto que preceda el primer separador de sección en el fichero de entrada se considerará parte del cuerpo, de manera que la utilidad nl tratará un fichero sin delimitadores de sección como una única sección de cuerpo.

Numerosas opciones pueden utilizarse para personalizar el resultado generado con esta utilidad, y algunas se detallan en la tabla 2-16.

Tabla 2-16 ; Opciones utilizadas con nl

Opción	***Uso***
-a	Numera todas las líneas
-t	Numera únicamente las líneas no vacías
-n	No numera las líneas. Es el valor por defecto de los encabezados y los pies de página.
-i NÚMERO	Incrementa el número de línea en una cantidad igual a NÚMERO. El valor por defecto es uno.
-p	No inicializa los números de línea al principio de cada página lógica.
-s CADENA	Añade la cadena de caracteres CADENA después de cada número de línea.
-v NÚMERO	Establece el NÚMERO inicial de cada página lógica.
-w NÚMERO	Especifica el NÚMERO de espacios que se reservan para los números de línea. El valor por defecto es seis.

A continuación se adjunta un ejemplo del uso de la utilidad nl, en el que se muestran los números correspondientes a todas las líneas.

```
$ nl prueba1
```

Usando el editor de flujo (sed)

La utilidad sed es un editor de flujo que puede tomar su entrada de un fichero o de datos que le son pasados desde una tubería (pipe). El sed normalmente opera globalmente sobre un fichero a no ser de que se le haya especificado lo contrario con símbolos de direccionamiento (adressing), en cuyo caso limitan el ámbito de ejecución del comando. Que el sed opera normalmente de forma global quiere decir que cada vez que encuentra el patrón especificado, lo reemplaza. El

direccionamiento puede ser usado para especificar el lugar donde se debe buscar para encontrar el patrón sobre el que actuar. Este direccionamiento puede especificar una sola línea o un grupo de ellas. Así mismo se puede excluir la actuación del comando sobre determinadas lineas usando el símbolo de exclamación invertido (!). Las expresiones regulares también pueden ser usadas para especificar lugares dentro de un archivo.

El sed puede ser usado para realizar simples substituciones o cambios mucho más complejos y potentes en un fichero. Las substituciones simples sobre un archivo se realizan usando la siguiente sintaxis:

```
sed -opción s/expresión_regular/substitución/flag nombre.archivo
```

El sed también funcionara con texto pasado desde la entrada standard y con texto de archivos especificados. En este caso, el archivo original quedará intacto y los cambios serán escritos en uno nuevo. Las "expresiones regulares" son una forma de buscar caracteres concretos y este punto se trata en la siguiente sección. En este caso el parámetro "s" le dice al sed que localice la expresión_regular que le hayamos dado y la borre, poniendo en su lugar lo que hayamos puesto como substitución.

Se pueden realizar múltiples substituciones usando el parámetro "-e". Por ejemplo:

```
$ sed -e 's/lisa/Lisa/' -e 's/nikki/Nikki' amigos.mios
$ sed -e 's/lisa/Lisa/' ; 's/nikki/Nikki' amigos.mios
```

Esta orden en cualquiera de sus dos sintaxis, buscaran la expresión regular "lisa" y la substituirían por "Lisa" y "nikki" por "Nikki" en el fichero amigos.mios .

También se puede utilizar el sed para ejecutarlo con una serie de ordenes y parámetros previamente definidos en un script, permitiendo automatizar y almacenar las opciones de uso mas frecuente y simplificar comandos demasiado largos. En este caso se debe de usar con la opción "-f" de la siguiente forma:

```
$ sed -f mi.script archivo.mio
```

Si se usa con un script, el script debe contener la sintaxis de los parámetros del sed con el mismo formato que usaríamos en un caso sin script: s/expresión-regular/substitución/flags

Las opciones de uso mas frecuente con sed son:

Opción	***Uso***
-V	Muestra información sobre la versión y sale.
-h	Muestra la información de ayuda del comando y sale.
-n	Evita que se envíe por pantalla el fichero modificado.
-e comando	Añade el comando a las ordenes a realizar.
-f script	Añade las ordenes de un script a las que se tienen que realizar.

Los flags se pueden añadir cuando se usa el comando "s" (ejemplos anteriores de substitución) de manera que permiten refinar y mejorar sus acciones. Estos son los flags que se pueden usar.

Flag	***Uso***
g	Aplica los cambios de forma global
p	Imprime por pantalla solo las líneas afectadas por algún cambio
Número	Reemplaza solo el caso del número (ordinal) expresado
w archivo	Crea un archivo solo conteniendo las líneas modificadas
I	Ignora las mayúsculas al realizar la búsqueda de la expresión

Un ejemplo del uso de de sed con un flag numérico sería el siguiente:

```
$ sed  's/hola/HOOOLA/5' archivo.mio
```

En este caso solo el 5º "hola" que encontrase en archivo.mio, sería reemplazado por "HOOOLA" y daría toda la salida por pantalla.

Así mismo, sed puede usar direcciones para expresar el lugar de los cambios en un fichero. Estas direcciones pueden ser números de línea, símbolos de numero de línea, o incluso expresiones regulares. Estas expresiones para la localización las vamos a ver un poco mas abajo de forma mas extensa.

Estas expresiones de localización preceden a los comandos y parámetros pasados a sed.

Un ejemplo sería este:

```
$ sed '4.11s/hombre/Varon/' archivo.mio
```

En este caso, solo se sustituiría la expresión "hombre" por "Varon" de las lineas 4ª a 11ª (ambas inclusive) en el fichero archivo.mio, y daría la salida por pantalla de todo el fichero y sus cambios.

Así mismo, podemos indicar que ignore determinadas líneas al hacer la substitución. Este caso le indicaría que NO la realizase en la línea 6ª del fichero archivo.mio

```
$ sed '!6s/hombre/Varon/' archivo.mio
```

Este tipo de direccionamiento y localización de líneas se usa normalmente con estos parámetros:

Parámetro	***Uso***
NUMERO	Solo realizara la substitución sobre la línea NUMERO
NUMERO.NUMERO	Realizara las substituciones en las líneas comprendidas entre los dos números ambos inclusive
$	Especifica que solo se realice sobre la última línea
!	Se realizará sobre todas las líneas excepto las expresadas tras !

Tema 103.3

Administración de archivos

Introducción

En este tema se tocarán temas referentes a la administración de los ficheros/directorios, se verá como copiar ficheros/directorios, moverlos, borrarlos, modificar sus propiedades, buscar archivos según su tamaño, tipo u hora.
Para terminar, se aprenderá como se usan los enlaces de ficheros y donde se encuentran generalmente muchos de los ficheros en un sistema Linux. El manejo de ficheros es la mayor parte del trabajo con sistemas Linux, por tanto es muy importante la compresión de los apartados tratados a continuación.

Los comandos que se verán en este tema son:

- cp
- find
- mkdir
- mv
- ls
- rm
- rmdir
- touch

Así mismo, se harán ejercicios sobre los mismos al final del tema, que serán muy parecidos a los realizados en los exámenes.

Este tema tiene un peso (importancia) de 3 de cara al examen final de la certificación LPI 101. El total de la suma de pesos de todos los temas es de 106.

Administrador de ficheros

Usar los comandos básicos de Unix para copiar y mover ficheros y directorios. Utilizar las opciones avanzadas del manejo de ficheros como la copia recursiva de múltiples ficheros y mover ficheros con un determinado patrón.

Cuando se trabaja con sistemas GNU/Linux, se necesita estar familiarizado con los comandos básicos del manejo de ficheros. En esta sección se cubrirán los más comunes, incluyendo los relacionados con el sistema de ficheros y los comandos para trabajar con directorios y ficheros.

Se necesitará saber para que se usa cada comando y las opciones más importantes de cara a la buena realización del examen.

Cuando se trabaja con ficheros en sistemas Linux, se ha tener en cuenta unas normas y restricciones. Los ficheros ocultos empiezan con un punto. Los nombres de ficheros pueden empezar con un número pero no pueden contener barras, signos de interrogación, asteriscos u otro tipo de caracteres reservados. La extensión de los ficheros no siempre es requerida, pero puede ser útil para un mantenimiento de los tipos de ficheros.

Las siguientes secciones examinan algunos de los comandos básicos cuando se trabaja con ficheros y directorios.

Cambiando directorios

Los directorios en un sistema GNU/Linux se distribuyen en una estructura de árbol. El directorio / , conocido como directorio raíz, contiene un número de directorios del sistema. Cada uno de estos directorios puede contener subdirectorios, como corresponde a un sistema de árbol. La figura 3-1 muestra esta estructura de directorios. El directorio del sistema y su localización son tratados en este capítulo.

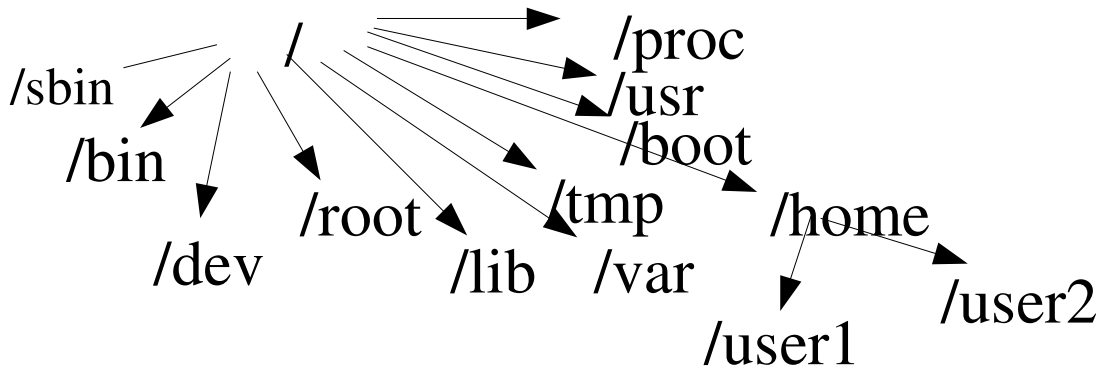

Figura 3-1: Estructura de directorios básica de Linux

El comando cd se utiliza para moverse por los directorios en un shell de Linux; cd viene de change directory. Este es uno de los comandos más simples usados en Linux. El comando pwd (present working directory) nos da el directorio actual, en el que se está en el momento de ejecutar el comando. Cuando se usa sin parámetros, el comando cd cambia del directorio de trabajo al directorio home. Es posible especificar también el directorio que quieres que sea tu directorio de

trabajo.
Ésta es la sintaxis del comando cd:

```
$ cd /directory
```

El comando cd se puede usar con el slash (/) cuando se quiere especificar un directorio absoluto, empezando desde el directorio raíz. Ésto permite moverse a cualquier parte especificando el directorio absoluto.
Sin el slash, cd buscará el directorio a partir de donde se ejecuta el comando (path relativo). A continuación se muestra un ejemplo del path relativo.

```
$ pwd
/home
$ cd angie
$ pwd
/home/angie
```

El comando cd se puede usar también con variables de entorno del sistema.
Los directorios "." y el ".." son ficheros ocultos que existen en cada directorio. El "." representa el directorio actual mientras que el ".." representa el directorio padre. Para el directorio / el ".." apunta a él mismo.
Otro símbolo utilizado es el carácter ~. Este carácter se usa para representar el directorio home del usuario.
Para terminar, el carácter – se pude usar para volver al directorio anterior.
Otro carácter especial que puede usarse con el comando cd es el *. Este carácter es usado para representar todos los caracteres.
El siguiente comando lista todos los ficheros de un directorio, que empiecen por la letra a, incluyendo los que solo sean una a.

```
$ ls a*
```

El siguiente ejemplo retrocede una vez en la estructura de directorios.

```
$ cd ..
```

Nota. Cuidado con el espacio entre el comando y los puntos. Este espacio es requerido en Linux, a no ser que creemos un alias para cd..

El siguiente es un ejemplo del uso de variables de entorno con el comando cd para cambiar del directorio donde estamos al directorio home del usuario. El símbolo $ se usa para especificar una variable de entorno en el comando.

```
$ cd $HOME
```

El comando cd no tiene opciones. Éste comando es uno de los más utilizados en Linux.

Listando los contenidos de un directorio

Otro comando muy utilizado en Linux es el comando ls, que se utiliza para ver el contenido de un directorio. Cuando el comando es utilizado sin opciones ni parámetros, lista todos los ficheros y directorios localizados en el directorio de trabajo que no tienen marca de archivos ocultos. Por defecto, se muestra en orden alfabético:

```
$ ls
```

El comando ls acepta paths como argumentos y entonces mostrará los ficheros y directorios de ese path. Se pueden usar paths absolutos:

```
$ ls /home/angie/stuff
```

También se pueden usar paths relativos:

```
$ ls stuff
```

El argumento puede contener cadenas de caracteres o caracteres especiales, en cuyo caso se mostrará un listado de todos los archivos y directorios que cumplan la condición.

```
$ ls a*
abc123names   abcnames   alphanames   alteredservices
```

Cuando se usa un path en conjunción con caracteres especiales, se muestra el path con el resultado:

```
$ ls /home/angie/stuff/n*
/home/angie/stuff/namelist   /home/angie/stuff/nicks

$ ls stuff/n*

stuff/namelist   /stuff/nicks
```

El comando ls pueden aceptar opciones como argumento. Hay un gran número de opciones para éste comando que permiten un gran control sobre el resultado. En la tabla siguiente se muestran las opciones más usadas:

Tabla 3-1: Opciones del comando ls

Opción	**Uso**
-a	Lista todos los contenidos del directorio.
-A	Trabaja como el -a excepto que no lista “.” y el “..”
-B	No se lista los ficheros que finalizan con ~
-d	Muestra el nombre del directorio en el listado
-L	Muestra la información para los ficheros enlaces o referenciales
-R	Muestra los directorios recursivamente.

Una variedad de opciones pueden ser usadas para especificar la información mostrada en un listado de ficheros. La tabla 3-2 cubre la información de las opciones de listado con el comando ls.

Tabla 3-2 Listado de las opciones de información de ls

Opción	***Uso***
-G	Especifica que grupo de información no se muestra
-I	Muestra el número inode
-l	Muestra el tipo de fichero, permisos, contador de enlaces permanentes, propietario, grupo propietario y fecha de la última modificación
-o	Muestra la misma información que -l menos la información de grupo que es excluida
-s	Muestra el tamaño del fichero en bloques de 1024 Kb

El comando ls utiliza a veces opciones para ordenar la salida, las cuales se muestran en la tabla 3-3

Tabla 3-3 Listado de las opciones de ordenación de la salida de ls

Opción	**Uso**
-c	Muestra el resultado acorde con la fecha de modificación o la fecha de modificación de inode
-f	Muestra el resultado con el orden en que han sido salvados en el directorio
-r	Muestra el listado en orden inverso
-S	Muestra el listado de acuerdo al tamaño, del más grande al más pequeño
-t	Muestra el listado de acuerdo a la fecha de modificación, mostrando primero el más reciente.
-u	Muestra el listado de acuerdo al último acceso, empezando por el más reciente.

La salida producida por el comando ls puede ser también controlada con otra serie de opciones. Estas opciones se muestran en la tabla 3-4.

Tabla 3-4 Listado de las opciones de apariencia de la salida de ls

Opción	**Uso**
-1	La salida se muestra una fila por linea
-C	La salida se muestra en columnas
-F	Muestra el nombre del fichero con una letra para especificar el tipo de fichero
-k	Muestra el tamaño del fichero en Kb
-m	Muestra los nombres de ficheros separados por comas
-n	Muestra el usuario y el número de grupo
-p	Muestra los nombres de ficheros con un carácter para especificar el tipo
-x	Muestra el nombre de fichero en columnas ordenadas horizontalmente
-T COLS	Asume que cada parada de tabulación está a cols columnas de ancho; el valor predeterminado es 8. ls emplea tabuladores donde es posible en la salida, por eficiencia. Si cols es cero, no usa tabuladores para nada.
-W COLS	Asume que la pantalla tiene cols columnas de ancho. El valor predeterminado se toma del controlador de terminal si es posible; si no, se emplea la variable de ambiente COLUMNS si está definida; de otro modo el valor predeterminado es 80

Todas éstas opciones se pueden combinar para crear una salida muy especifica usando el comando

ls. Los siguientes son unos ejemplos de como éstas opciones se pueden usar para controlar que datos se muestran y como se muestran. A través de las tablas anteriores se puede comprobar que opciones se usan.

Saber usar el comando ls, junto a sus posibles opciones, es muy importante de cara al examen.

El siguiente ejemplo lista los nombres de ficheros de acuerdo con la fecha del último acceso, con la / para mostrar los directorios:

```
$ ls -pu
morestuff/      nicks  list      readmes        alphanames    namelist
```

El siguiente ejemplo muestra el contenido del directorio /home/angie/stuff y todos sus subdirectorios:

```
$ ls -R /home/angie/stuff
```

A través de los ejemplos mostrados, se puede observar que el comando ls puede ser una potente herramienta para sacar información de los ficheros y directorios. El comando se usa muchas veces en conjunción con otros comandos y ficheros usando filtros y redireccionamientos. Ésto permite salvar la salida a un fichero o procesarlo con otras utilidades.

Determinando el tipo de fichero

El comando ls provee de mucha información cuando se examinan ficheros, pero no muestra información sobre el tipo de contenido de los mismos.
El comando file puede ser usado para aprender más sobre el tipo contenido de los ficheros en un sistema Linux. La salida del comando file incluye una de las siguientes palabras: text, executable, data o directory. Este comando acepta argumentos para especificar que ficheros examina. Se pueden usar una serie de opciones con este comando, como se muestran en la tabla 3-5

Tabla 3-5 Opciones usadas con file

Opción	Uso
-b	Especifica que el nombre de fichero no se muestre en la salida.
-f <nombrefichero>	Especifica que el fichero <nombrefichero> contiene los nombres de los ficheros a examinar.
-n	Muestra el resultado después del chequeo. Esto puede ser útil cuando trabajamos con una serie de ficheros que van a ser enviados a otro comando.
-v	Muestra la versión
-z	Intenta examinar el contenido de los ficheros comprimidos.

Algunos ejemplos del comando file y sus opciones.
En el primer ejemplo, el * hace que se muestre el tipo de contenido de todos los archivos del directorio actual.

```
$ file *
```

En el siguiente ejemplo, el nombre de fichero no es incluido dentro del listado.

```
$ file -b /etc/lilo.conf
```

En el ejemplo siguiente, el comando file se usa para examinar el tipo de contenido de un fichero comprimido:

```
$ file -z /usr/info/tar.info.gz
```

Cambiando la fecha de acceso (time stamp)

El comando touch permite cambiar la fecha de último acceso de un fichero. Si se especifica el nombre de fichero, pero el fichero no existe, se crea un fichero vacío con ese nombre.
Las opciones del comando touch se pueden ver en la tabla 3-6.

Tabla 3-6 Opciones usadas con touch

Opción	Uso
-a	Solo cambia la fecha de acceso en el fichero
-c	No crea un fichero si el nombre no existe.
-d <string>	Usado para especificar el formato de la fecha en el fichero usando la fecha actual.
-r <file>	Utiliza la fecha de <file> en vez de la del sistema
-t <TIME>	Utiliza una fecha como argumento en vez de la del sistema.

El comando touch permite usar la fecha actual por defecto o especificar la fecha de último acceso. El comando se usa a menudo para crear ficheros vacíos con un nombre especifico. Esto puede ser útil cuando se ejecuta un script que necesita un fichero que no existe.

El sistema de rotado de los ficheros de log, es un ejemplo de uso del comando touch. En dicho script, se renombra el fichero de log actual a una version más antigua. Tras el renombrado, con el comando touch, se crea el fichero que seguirá almacenando los logs:

```
...
mv syslog syslog.1          -> renombramos el fichero
touch syslog                -> se crea de nuevo el fichero, vacío
...
```

El siguiente ejemplo muestra un script en el cual el comando touch se usa para crear un fichero vacío.

```
$ cp mylogs stuff/mylogs.`date +%m.%d.%y`
$ rm -f mylogs
$ touch mylogs
```

Copia el contenido de mylogs al directorio stuff con el nombre mylog.mes.dia.año (por ejemplo mylogs.12.11.00) El fichero original mylog es borrado y entonces se usa el comando touch para crearlo de nuevo vacío.
Uno de los usos habituales del comando touch, es la creación de archivos vacios.

Copiando Archivos

Cuando se trabaja con archivos en cualquier sistema, generalmente se necesita copiar archivos. Linux incluye dos comandos para copiar archivos.
El comando cp se usa para copiar archivos y directorios. Es el comando estándar para copiar archivos de una localización del sistema a otra.
Cuando se quiere copiar archivos de un formato de archivo a otro, se hará uso el comando dd.

El comando cp (abreviatura de copy) se utiliza para la copia estándar de archivos en sistemas Linux. Este comando se usa para crear una copia nueva e independiente del archivo o directorio original. Se pueden usar muchas opciones con el comando cp para modificar las copias creadas. Las opciones se resumen en la tabla 3-7:

Tabla 3-7 Opciones del comando cp

Opción	Uso
-a	Especifica que los enlaces y atributos del archivo original deben ser transferidos a la nueva copia.
-d	Especifica que los enlaces se deben mantener cuando se copia.
-f	Sobreescribe cualquier archivo de destino existente.
-i	Pregunta antes de sobreescribir cualquier archivo de destino existente.
-l	Especifica que se creen enlaces fuertes (ver más adelante en este capítulo) en lugar de copias del archivo.
-p	Mantiene el propietario, grupo, permisos y timestamp del archivo original.
-r	Copia directorios y sus contenidos de forma recursiva mientras copia cada archivo como archivo estándar. Esta opción no podrá ser usada con algunos archivos especiales.
-R	Copia directorios y sus contenidos de forma recursiva, manteniendo los directorios.
-s	Crea enlaces simbólicos (ver más adelante) de los archivos que no sean directorios.
-v	Muestra todos los nombres de los archivos según se van copiando.

Estas opciones se pueden combinar cuando se copian archivos. Junto con las opciones, también se usan argumentos con el comando cp. La sintaxis correcta del comando es la siguiente:

```
cp -opciones origen destino
```

Cuando el destino especificado es un directorio, el archivo de origen se copia a ese directorio con el mismo nombre que el archivo original. Cuando el destino especificado no es un directorio, el archivo de origen se copia a la localización especificada con el nombre del destino.
Los siguientes son algunos ejemplos de uso del comando cp.
En el primer ejemplo, el archivo “marital” se copia al directorio “stuff”.

```
$ cp marital stuff
```

En el segundo ejemplo, el directorio “/home/angie/stuff” y su contenido se copia al directorio “/home/angie/otrostuff”. Esto es particularmente útil, ya que permite copiar el contenido entero de un directorio usando un único comando:

```
$ cp -r /home/angie/stuff /home/angie/otrostuff
```

Pregunta de Examen: Ésta es una opción importante que aparece frecuentemente como pregunta de examen.

Copiando y convirtiendo archivos con diferente formato

El comando dd (abreviatura de Direct Dump) se usa para copiar y convertir archivos de un formato a otro simultáneamente. Este comando tiene diferentes opciones y diferente sintaxis que el comando cp. La sintaxis utilizada para el comando dd es la siguiente:

dd [opciones]

El comando dd, por defecto, escribe datos desde la entrada estándar hacia la salida estándar. Las opciones se utilizan para cambiar estos valores por defecto. Las opciones para el comando dd se muestran en la tabla 3-8:

Tabla 3-8 Opciones del comando dd

Opción	Uso
if=FILE	Especifica la localización del archivo de origen para ser utilizado en lugar de la entrada estándar. Éste es el archivo de origen.
of=FILE	Especifica la localización del archivo de destino para ser utilizado en lugar de la salida estándar. Éste es el archivo de destino.
ibs=BYTES	Especifica el número de bytes leídos de cada vez.
obs=BYTES	Especifica el número de bytes escritos de cada vez.
bs=BYTES	Especifica el número de bytes a escribir y leer de cada vez.
cbs=BYTES	Especifica el número de bytes a convertir de cada vez.
skips=BLOCKS	Especifica los bloques a saltar en el archivo de origen antes de empezar a copiar.
seek=BLOCKS	Especifica los bloques a saltar en el archivo de destino antes de empezar a escribir.
count=BLOCKS	Especifica los bloques a copiar del archivo de origen en lugar de copiar el archivo completo.

El comando dd se puede utilizar para variedad de tareas especiales.
Por ejemplo:

```
$ /bin/dd if=/dev/hda5 bs=512 count=1 of=/mnt/win_c/bootsek.lin
```

Esto creará un archivo llamado bootsek.lin en la partición montada en /mnt. El archivo se escribirá como un bloque con un tamaño de 512 bytes.
En este ejemplo, el comando dd se utiliza para copiar un archivo desde una partición a otra partición formateada con el sistema de archivos FAT. Este comando es útil también para trabajar con sistemas de copia de seguridad de tipo cinta.

Moviendo Archivos

Se puede mover un archivo manualmente copiándolo a la nueva localización y borrando luego el archivo original. Sin embargo, Linux incluye un comando para mover archivos que automatiza esta tarea. El comando mv (abreviatura de move) permite mover y renombrar archivos en sistemas Linux. Este comando funciona como el comando cp, utilizando la misma sintaxis. Las opciones para el comando mv son algo diferentes y se resumen en la tabla 3-9:

Tabla 3-9 - Opciones del comando mv

Opción	Uso
-f	Borra los archivos existentes sin pedir confirmación.
-i	Pide confirmación al usuario antes de sobreescribir archivos.
-u	Especifica que los archivos no serán movidos al destino si tienen fecha de modificación igual o más reciente.
-v	Muestra por pantalla los archivos movidos.

En el siguiente ejemplo, todos los archivos que cumplan el patrón, empezando por "archivo" serán movidos al directorio "misarchivos" y mostrará por pantalla los nombres de los archivos movidos:

```
$ mv -v archivo* misarchivos
```

En el segundo ejemplo, el directorio "misarchivos" se renombra a "archivos":

```
$ mv misarchivos archivos
```

Borrado de archivos

Otra tarea frecuentemente necesaria cuando trabajamos con archivos y directorios es la eliminación de los mismos. El comando rm (abreviatura de remove) se utiliza para borrar archivos y directorios en sistemas Linux. Utiliza la siguiente sintaxis:

```
rm -opciones ARCHIVO
```

Se pueden utilizar muchas opciones con el comando rm. Las utilizadas más frecuentemente son las que se muestran en la tabla 3-10:

Tabla 3-10 - Opciones del comando rm

Opción	Uso
-d	Utilizada por el superusuario. Elimina directorios sin tener en cuenta si están vacíos.
-f	Ejecuta el comando sin pedir confirmación, incluso si los archivos especificados no existen.
-i	Pide confirmación al usuario para eliminar los archivos.
-r	Elimina el contenido del directorio de forma recursiva.
-v	Elimina archivos y muestra por pantalla los nombres de los archivos eliminados.

En el siguiente ejemplo se borrarán todos los archivos del directorio actual (pwd) cuyo nombre

comience con “nn”:

```
$ rm nn*
```

El segundo ejemplo eliminará todos los archivos del directorio “archivos” de forma recursiva, así como el propio directorio y muestra por pantalla los nombres de los archivos eliminados:

```
$ rm -frv archivos
removing archivos/archivo1
removing archivos/archivo2
removing archivos/archivo3
removing archivos/archivo4
removing archivos/archivo5
removing the directory itself: archivos
```

Creando directorios

En todo este apartado se ha tratado el tema de la creación y eliminación de archivos, así como de la eliminación de directorios. Ahora trabajaremos con el comando *mkdir* (abreviatura de make directory), que se utiliza para crear directorios. Es un comando muy simple, que creará un directorio cada vez. Cuando se utiliza con la opción -p, se crearán también los directorios padres si es necesario.

Para crear los directorios:
/home/angie/nuestrosarchivos,
/home/angie/nuestrosarchivos/misarchivos y
/home/angie/nuestrosarchivos/tusarchivos
utilizaremos el siguiente comando:

```
$ mkdir /home/angie/nuestrosarchivos \
/home/angie/nuestrosarchivos/misarchivos \
/home/angie/nuestrosarchivos/tusarchivos
```

En el comando escrito, el directorio “nuestrosarchivos” se crea primero. Entonces el sistema crea los directorios “misarchivos” y “tusarchivos” sin el directorio “nuestrosarchivos”.

Entendiendo la jerarquía del sistema de ficheros

El sistema de ficheros de Linux sigue una estructura en forma de árbol. La raíz (/) contiene los directorios principales. Cada uno de los directorios contenidos en el directorio raíz puede, a su vez, contener más directorios. Es importante familiarizarse con algunos directorios clave del sistema de ficheros, así como entender que función llevan a cabo. En esta sección nos centraremos en algunos de estos directorios clave.

Localización de los directorios estándar

En los siguientes directorios se encuentran algunos de los ficheros estándar de un sistema Linux:

-/etc : Contiene muchos de los ficheros de configuración y guiones (scripts) del sistema, así como ficheros de configuración propios de algunos programas.
-/etc/skel : Contiene los ficheros que se copian en el directorio del usuario cuando éste es creado.
-/usr : Contiene los subdirectorios donde se almacenan las aplicaciones y ficheros fuente usados por los usuarios, así como también librerías usadas por los programas. Hay varios de estos subdirectorios destacables:
    -/usr/bin : Aquí viven los programas de las diversas aplicaciones del sistema. Cada usuario debería tener este subdirectorio en su variable de PATH.
    -/usr/sbin : Aquí viven los programas usados por el administrador (o superusuario).
    -/usr/local : Aquí se instalan por defecto las aplicaciones que no forman parte del sistema operativo. En concreto cabe destacar las siguientes:
        -/usr/local/bin : Contiene los ejecutables de dichas aplicaciones.
        -/usr/local/sbin : Contiene los ejecutables de dichas aplicaciones que solo puede ejecutar el administrador.
-/var/log : Contiene los ficheros de traza o log del sistema.
-/var/spool : Contiene la bandeja de correo y la cola de impresión, con los correspondientes ficheros de correo electrónico de cada usuario además de los ficheros que hubieran mandado a imprimir.
-/bin : Aquí viven los programas ejecutables más importantes del sistema operativo. También se encuentran los binarios usados durante el inicio del sistema.
-/sbin : Contiene los programas ejecutables usados por el administrador.

Directorios del sistema

Algunos de los directorios con propósitos específicos que deberíamos conocer son los siguientes:

-/ : Es el directorio raíz (de root, en inglés). De él cuelgan el resto de subdirectorios del sistema.
-/root : Es el directorio personal del administrador.
-/home : Es el directorio que contiene los directorios personales de los usuarios.
-/boot : Contiene los ficheros usados por el kernel durante el arranque.
-/dev : Contiene los ficheros de acceso a los dispositivos configurados en el sistema.
-/proc : proc es un directorio virtual (esto significa que sólo existe en tiempo de ejecución) usado por el núcleo del sistema (kernel) para comunicarse y almacenar ficheros con información del sistema.

Localización de ficheros

Tarde o temprano va a ser necesario encontrar algún fichero o programa. Para ello Linux proporciona unas cuantas herramientas:

find
Se usa para buscar ficheros. Find empieza en un directorio y luego desciende por sus subdirectorios

buscando el patrón indicado. Si no se especifica ningún directorio de búsqueda, *find* recoge el valor de la variable PWD, es decir, asume el directorio actual. Su sintaxis es como sigue:

find /ruta expresión

locate

Locate es una forma segura de indexar ficheros en el sistema para poder buscarlos más rápidamente en un futuro. Para ello guarda rutas -además de permisos y características de pertenencia, para preservar la privacidad- en una base de datos llamada slocate. Su sintaxis es como sigue:

locate -opciones argumentos

which

El comando which devolverá por la salida estándar aquello que ejecutaría la shell (dónde se encuentra el comando). Devolverá el comando con la ruta completa. La búsqueda la realiza en las rutas encontradas en el PATH. El comando *which* es útil para verificar que el comando que se quiere ejecutar es realmente el que se desea. Su sintaxis es como sigue:

which -opciones argumentos

whereis

Busca ficheros fuente, binarios y secciones de páginas de manual para los comandos especificados como argumentos. A esos argumentos se les quita los segmentos de ruta que pudieran llevar, los prefijos y las extensiones. Las búsquedas se realizan en una lista de rutas estándar.

Tema 103.4

Usando streams, pipes y redirecciones

Introducción

En este capítulo se verá como usar Streams, Tuberías (pipes) y Redirecciones Unix. Conectar archivos a comandos y comandos a otros comandos para procesar datos de texto de forma eficaz. Incluye la redirección de la entrada estándar, salida estándar y error estándar; y canalizar la salida de un comando como entrada de otro comando o como argumento (usando xargs); enviando la salida hacia stdout (salida estándar) y a un archivo (usando tee).

Los comandos que se verán en este tema son:

tee
xargs
<
<<
>
>>
|

Así mismo se harán ejercicios sobre los mismos al final del tema, que serán muy parecidos a los realizados en los exámenes.

Este tema tiene un peso (importancia) de 5 de cara al examen final de la certificación LPI 101. El total de la suma de pesos de todos los temas es de 106.

Uso de los Streams (flujos de datos), Pipes (tuberías), y Redireccionamientos

Una de las muchas ventajas de los sistemas Linux y Unix es la noción de que cualquier cosa es un fichero. Un disco y sus particiones, cintas, terminales (pantallas), puertos serie, el ratón, e incluso el audio están mapeados (organizados y jerarquizados) en el sistema de ficheros. Este mapeo permite a los programas interactuar con cualquier tipo de dispositivos y ficheros del mismo modo, lo cual simplifica su interfaz enormemente.

Cada dispositivo que está representado por un fichero es llamado dispositivo de fichero (device file o dev), lo cual lo convierte en un objeto especial en el sistema de ficheros que proporciona un interfaz hacia el dispositivo. El kernel asocia los drivers del dispositivo (el físico) con varios dispositivos de ficheros, así es como el sistema da la apariencia de que a los dispositivos se puede acceder como si fuesen ficheros. Utilizando una terminal (pantalla) como ejemplo, un programa lee del dispositivo fichero del terminal que recibirá los caracteres tecleados en el teclado. La escritura en el terminal (fichero) provoca que los caracteres aparezcan en pantalla. Aunque parece raro pensar en un terminal (pantalla) como un fichero, este mismo concepto unifica y dota de simplicidad la programación de Linux y Unix.

Entrada y salida estándar and File Descriptors por defecto

La entrada/salida estándar (standard I/O) es responsabilidad del shell, habitualmente se usan en todas las utilidades basadas en consola de GNU/Linux para controlar y dirigir las entradas de programa, las salidas y la información de errores. Cuando un programa (proceso) se ejecuta tiene disponibles tres *streams*, o tipos de flujo de datos: uno para la entrada, otro para la salida y otro para mostrar mensajes de error o de diagnóstico, se describen a continuación:

Entrada estándar (abreviada *stdin*)
Este "file descriptor" es una entrada de flujo de texto. Por defecto está asociada al teclado. Cuando se teclea en un programa interactivo de texto, se está alimentando la entrada estándar. Como se puede ver, algunos programas emplean uno o más nombres de ficheros como argumentos de la línea de comandos e ignoran la entrada estándar. La entrada estándar está asignada al *file descriptor 0*.

Salida estándar (abreviada *stdout*)
Este "file descriptor" es la salida por defecto del flujo de caracteres de un programa. Por defecto está asociada al terminal (o ventana). La salida generada a través de los comandos es escrita en la salida estándar para ser reflejada en la pantalla. La salida estándar está asignada al *file descriptor 1*.

Error estándar (abreviada *stderr*)
Este "file descriptor" es también una salida de flujo de caracteres, pero se usa exclusivamente para mostrar errores u otra información no relacionada con los resultados normales del comando. Por defecto esta salida está asociada al terminal de salida, igual que la salida estándar. Ello implica que tanto stdout como stderr convergen en la pantalla, de modo que pueden ser confundidas. El modo de controlar esto se tratará más adelante. La salida de error estándar está asociada al *file descriptor 2*.

Los descriptores de fichero de la entrada/salida estándar se usan en el momento de creación y posterior ejecución de un programa para leer y escribir ficheros del disco. Ello permite asociar

comandos a ficheros y a dispositivos, gestionando los comandos de entrada y salida exactamente como se desee. La diferencia es que los programas ya tienen los descriptores de fichero -asignados por defecto gracias a la shell- y no es necesario crearlos explícitamente.

Los datos enviados a las utilidades que se verán en este capítulo suelen venir tanto de la stdin como de un archivo. Muchas utilidades escriben la salida y los errores hacia la stdout y stderr respectivamente. Esto es adecuado para ver los resultados; sin embargo, si se desea guardar los resultados en un archivo al que se pueda acceder más tarde, Linux tiene una solución para ello: la Redirección.

Redirecciones

Es posible cambiar la stdin, stdout y stderr cuando se trabaja con una shell de Linux. El stdin se redirige por medio del símbolo <. Muchas utilidades trabajan usando información proveniente desde el stdin o desde un archivo, pero también se pueden enviar datos a esas aplicaciones usando las redirecciones.
Un ejemplo del uso de la redirección de stdin es el siguiente, en el que el comando sort, recibe la entrada de datos desde un fichero 'listaNombres' en lugar de hacerlo por teclado (entrada estándar):

```
$ sort < listaNombres
```

Mucho más a menudo se querrá redirigir la stdout. Esto se hace por medio del símbolo >.
Ver el contenido de un archivo en orden alfabético puede no tener tanto valor como poder guardar esa lista ordenada para futuras consultas. Usando la redirección se puede guardar la salida en otro archivo.
El siguiente ejemplo usa la redirección para enviar el resultado del comando sort a un archivo llamado “alfaNombres”:

```
$ sort listaNombres > alfaNombres
```

Además de redirigir stdout para crear un archivo nuevo, se puede querer redirigir stdout para añadir datos a un archivo ya existente. Esto se hace por medio del símbolo >>.
En el siguiente ejemplo, el contenido del archivo “nicks” se ordena y el resultado se añade al archivo “alfaNombres”:

```
$ sort nicks >> alfaNombres
```

Es importante remarcar que cuando se crean ficheros, los operadores de redirección de salida son interpretados por el shell después de que los comandos se hayan ejecutado. Ello implica que ningún fichero creado con un comando de redirección puede ser abierto antes. Por ello, no se puede modificar un fichero según la siguiente secuencia:

```
$ grep "stuff" file1 > file1  iiiiiNO hacerlo!!!!!
```

Si file1 contiene algo importante, este comando será desastroso ya que un file1 vacío sobreescribirá el original. El comando grep será el último en ejecutarse, resultando de esto una total pérdida de datos del fichero original file1 ya que el fichero que lo reemplaza está vacío. Para evitar este problema, sencillamente se debe usar un fichero intermedio y renombrarlo:

```
$ grep "stuff" file1 > file2
```

```
$ mv file2 file1
```

También se puede redirigir stderr junto con stdout. Para redirigir stderr y stdout hacia un archivo nuevo, se usa el símbolo >&. Por ejemplo:

```
$ sort listaNombres >& listaNombresOrdenada
```

Si no hay errores, el contenido del archivo de destino será el mismo que si no se hubiera redirigido stderr. Sin embargo, usando >&, cualquier error que ocurra será incluido en el archivo.

Si se quiere redirigir únicamente stderr hacia un archivo, se puede usar el símbolo 2>. Por ejemplo, para redirigir sólo los errores del comando sort usado antes, se usará:

```
$ sort listaNombres 2> erroresListaNombres
```

Esto también es útil si se quiere ejecutar un comando mientras se redirige stderr y stdout hacia dos archivos separados. El ejemplo siguiente enviará stderr hacia el archivo "erroresLista" y stdout hacia el archivo "listaOrdenada":

```
$ sort listaNombres 2> erroresLista > listaOrdenada
```

Otro tipo de redirección es la llamada "herefile"(<<) la cual consiste en especificar la entrada a un comando en varias lineas después de la llamada al mismo, terminando con un valor centinela, se ve más fácil con un ejemplo:

Si se ejecuta por consola el siguiente comando:

```
$ sort << quietoparado
```

No saldrá el prompt PS2 (>) en el cual se pondrá la lista de cosas que se quieren ordenar:

```
>hola
>caracola
>zarpando
```

para terminar de meter el listado, se insertará la palabra quietoparado (puede ser cualquier otra indicada después del símbolo <<)

```
>quietoparado
```

inmediatamente después saldrá el resultado del comando:

```
>caracola
>hola
>zarpando
```

y devolverá al prompt $

En la tabla 4-1 se listan las redirecciones más comunes de I/O estandar para el bash shell, especificadas en los Objetivos del LPI.

Las sintaxis de la redirección puede ser significativamente distinta si se usa otro shell.

Tabla 4-1. Redirecciones de I/O estandar para el bash shell

Función de Redirección	Sintaxis de bash
Enviar stdout a file	$ cmd > file $cmd 1> file
Enviar stderr a file	$ cmd 2> file
Enviar stdout y stderr a file	$ cmd > file 2>&1
Enviar stdout a file1 y stderr a file2	$ cmd > file1 2> file2
Recibir por stdin de file	$ cmd < file
Recibir por stdin desde teclado en varias lineas	$ cmd <<valor_de_parada
Añadir stdout a file	$ cmd >> file $ cmd 1>> file
Añadir stderr a file	$ cmd 2>> file
Añadir stdout y stderr a file	$ cmd >> file 2>&1

En el Exámen:
Se ha de estar preparado para diferenciar entre nombres de ficheros y nombres de comandos en aquellos comandos donde se utilizan operadores de redirección. También es necesario entender la sintaxis en comandos con redirección para estar seguro sobre que comando o fichero es fuente de datos y cual de ellos es destino.

Tuberías (pipes):

Junto con las redirecciones, se puede querer usar otros comandos. Esto se puede hacer por medio del comando “pipe” (|). Esto permite usar una línea de comandos para enviar los datos de salida de un comando como datos de entrada de otro comando.
En el siguiente ejemplo, el archivo listaNombres se ordena alfabéticamente, y después se añaden números de línea a cada nombre usando el comando nl:

```
$ sort +1 listaNombres | nl
```

Estos datos también se pueden canalizar hacia otra utilidad o redirigir hacia un archivo:

```
$ sort +1 listaNombres | nl > nombresNumerados
```

Este ejemplo muestra como se pueden usar juntas las tuberías y las redirecciones para procesar datos utilizando distintas utilidades y después guardar el resultado en un archivo.
Así como stdout se puede redirigir para guardar el resultado en un archivo, también se puede incluir stderr utilizando el símbolo |&.

Por ejemplo, el siguiente comando ordena el contenido de “listaNombres”, añade números de línea, y añade cualquier información de error:

```
$ sort +1 listaNombres |& nl
```

Otro ejemplo:

```
$ grep "01523" order* | less
```

Se buscan en los ficheros cuyo nombre empiecen por “order” las líneas que contienen la palabra “01523”. Creando este pipe, la salida estándar del *grep* es enviada a la entrada estándar del *less*. El mecanismo de esta operación es tratado por el shell y transparente al usuario.

Los pipes pueden ser usados con una serie de muchos comandos. Cuando más de dos comandos se unen, la operación resultante es conocida como pipeline o text stream (flujo de texto), implicando esto que el flujo de texto pasa de un comando al siguiente.

Bifurcaciones (tee)

La utilidad tee copia la entrada estándar hacia la salida estándar y también hacia otros archivos dados como argumentos, de modo que stdin aparece tanto en stdout como en los archivos. Esto es útil cuando se quiere enviar datos no sólo hacia una tubería, sino también guardar una copia en un archivo. Si el archivo que se está escribiendo no existe todavía, se crea. Si el archivo ya existe, se sobreescribe el contenido del mismo, a no ser que se utilice la opción de añadir.
Las opciones del comando tee son las siguientes:

-a	Añade la entrada estándar la archivo especificado
-f	Ignora las señales de interrupción que podrían ser utilizadas para parar o reiniciar el proceso.

En el siguiente ejemplo, el archivo “listaNombres” se ordena primero alfabéticamente y después se canaliza hacia tee de forma que la salida se guarda también en el archivo abcNombres. Los datos son enviados entonces al comando nl, de forma que se numeran las líneas, y la salida es entonces enviada a otro archivo llamado abc123Nombres:

```
$ sort +1 listaNombres | tee abcNombres | nl > abc123Nombres
```

Pasando multiples argumentos a otros comandos (xargs)

La utilidad xargs se utiliza para pasar un gran número de argumentos a otros comandos. La utilidad args lee argumentos desde la entrada estándar, delimitados por espacios en blanco (el espacio en blanco funciona como un carácter normal cuando se utiliza entre comillas simples o dobles o tras una barra invertida) o caracteres de salto de línea, y ejecuta el comando una o más veces con cualquier número de argumentos iniciales seguidos por los argumentos leídos desde la entrada estándar. Las líneas en blanco de la entrada estándar se ignoran. Esto permite a un comando procesar más argumentos de los que en condiciones normales podría manejar.

En el ejemplo siguiente se buscan todos los archivos “README” en la base de datos “locate”, usando el link locate. Usando xargs, los nombres de los archivos son enviados a la utilidad cat, que mostrará el contenido de los mismos en pantalla. Todo el texto de estos archivos se envía a la utilidad fmt, que formatea los datos hasta un máximo de 60 caracteres por línea. La salida se envía finalmente al archivo /home/angie/readmes:

```
$ locate README | xargs cat | fmt -60 > /home/angie/readmes
```

Tema 103.5

Creando, monitorizando y matando procesos

Introducción

En éste capítulo veremos como administrar los procesos. Esto incluye saber ejecutar procesos en primer y segundo plano, cambiarlos de plano, monitorizar procesos actuales, ordenarlos según varios parámetros, enviar señales a los procesos y matar procesos innecesarios para el sistema/usuario.

Los comandos que se verán en este tema son:

- &
- bg
- fg
- jobs
- kill
- nohup
- ps
- top

Así mismo se harán ejercicios sobre los mismos al final del tema, que serán muy parecidos a los realizados en los exámenes.

Este tema tiene un peso (importancia) de 5 de cara al examen final de la certificación LPI 101. El total de la suma de pesos de todos los temas es de 106.

Que es un proceso

Un proceso es, en resumen, un programa en ejecución.

Todos los comandos ejecutados a lo largo del curso acaban de una forma u otra, por generar un proceso que realiza la tarea. La shell que usamos para interactuar con el sistema también es un proceso. De hecho, la shell es un buen ejemplo para entender las relaciones de parentesco entre procesos.

Cuando se ejecuta un comando en la shell, primero comprueba si es uno de sus comandos internos. Por ejemplo export:

```
$ export
```

export es un comando interno de la shell, no necesita la ejecución de ninguna utilidad externa a la misma shell para llevar a cabo la tarea.
Otro ejemplo, el comando que nos muestra la fecha:

```
$date
```

En caso de no ser un comando interno, la shell ejecuta el comando necesario para llevar a cabo la tarea, invocando un nuevo proceso. Este nuevo proceso se le conoce como proceso hijo (en relación a nuestra shell) y la propia shell es el proceso padre (de ese proceso hijo).

Cuando el proceso ha acabado de realizar su tarea, devuelve los resultados al proceso padre y finaliza su ejecución. El proceso hijo puede necesitar invocar otro proceso donde delegar parte de la tarea, convirtiéndose a su vez, en el proceso padre del proceso recién invocado. En los sistemas multitarea como GNU/Linux, los procesos pueden llegar a desarrollar auténticos árboles genealógicos, ya que los procesos pueden se padres e hijos y además, varios procesos pueden ser hijos del mismo proceso padre a la vez.

Los demonios

De los diferentes procesos que pueden estar ejecutándose en un sistema GNU/Linux, esto es, procesos invocados por el usuario, procesos invocados por otros usuarios y procesos invocados por el sistema operativo, los últimos son los más especiales y se les conoce como demonios (daemons).

Los demonios son procesos que el sistema operativo invoca para proporcionar servicios. Una característica de los demonios es que no suelen interactuar mucho. Proporcionan la funcionalidad para la que están programados de manera silenciosa, al menos en cierta forma, puesto que la gran mayoría utilizan ficheros de log para registrar sus transacciones. Entre los demonios más comunes están los de impresión, los de correo, los de periodicidad de tareas y monitorización. Pero hay muchos más.

Trabajando con ps

El comando ps da información sobre los procesos corriendo sobre el sistema. Invocado de la manera más sencilla:

```
$ ps
```

Devuelve un listado con los procesos que se lancen con el usuario actual y que aún se están ejecutando. En la última línea aparecerá el proceso que representa a *ps*, ya que es el último proceso lanzado.

```
  PID TTY          TIME CMD
18621 pts/24   00:00:00 bash
18628 pts/24   00:00:00 ps
```

Echando un vistazo al resultado:

- La primera columna, marcada como PID, representa el identificador de proceso (Process ID, PID). Este identificador de proceso se asigna desde cero hasta un límite marcado por el propio sistema operativo. Cuando el identificador llega a ese límite, vuelven a asignarse los identificadores libres empezando otra vez desde cero.
- La segunda columna indica la terminal asociada al proceso. Hay procesos que no tienen asignada terminal (demonios...), estos aparecen marcados con un signo de interrogación (?) en esta columna.
- La tercera columna indica el porcentaje de tiempo de procesador que el proceso está usando. Normalmente, los procesos ejecutan sus tareas muy rápidamente, durante intervalos cortos. Luego se mantienen a la espera. Un proceso que muestre lecturas altas en esta columna puede repercutir en el rendimiento del resto del sistema.
- La cuarta columna es el nombre del proceso (el nombre del comando). La primera línea es siempre la shell sobre la que se ejecutan los procesos. A esta shell (de hecho al primer proceso del usuario) se le llama líder de sesión (session leader).

A ps se le llama frecuentemente de las siguientes formas:

```
$ ps -a
```

Devuelve el listado de todos los procesos sin incluir al líder de sesión ni los procesos sin terminal asociada.

```
$ ps -e
```

Devuelve el listado de todos los procesos. La opción -e es sinónimo de la opción -A.

```
$ ps -l
```

Devuelve un listado extendido. Con información sobre UID's (identificadores de usuario o User ID's, marcado como UID), los PID's del proceso padre (marcado como PPID), información sobre si el proceso entra en las expectativas de planificación de procesos del kernel (marcado como C), o la fecha de inicio del proceso (marcado como TIME). Hay que destacar que en este formato, el nombre del proceso se sustituye por la cadena entera entrada con la que se lanzó el proceso, con parámetros, opciones y argumentos (marcado como CMD).

Por último:

```
$ ps -u root
```

Devolverá todos los procesos que ha lanzado el usuario especificado tras el parámetro -u (en el ejemplo debe sacar información sobre los procesos invocados por root o por el sistema).

Trabajando con *pstree* y *top*

Existen dos comandos relacionados con el comando *ps* que ofrecen una vista de los procesos ligeramente diferente a como lo hace *ps*.

El primero de ellos es *pstree*, que ofrece una visualización gráfica (aunque en modo texto) de las relaciones que existen entre los procesos:

```
$ pstree
init-+-amd
     |-atd
     |-cron
     |-6*[getty]
     |-httpd.apache—2*[httpd.apache]
     |-inetd
     |-kdewizard
     |-kflushd
     |-klogd
     |-kpiod
     |-kswapd
     |-kwmsound
     |-lockd
     |-mdrecoveryd
     |-rpc.portmap
     |-rpc.rstatd |-rpciod
     |-sendmail
     |-sh—rc.gui—kdm-+-X
     |                '-kdm—kwm-+-kaudioserver
     |                          |-kbgndwm
     |                          |-kfm—kvt—bash-+-awk
     |                          |              |-cat
     |                          |              |-paste
     |                          |              |-pstree
     |                          |              '-vi
     |                          |-kpanel
     |                          |-krootwm
     |                          '-kwmsound
     |-syslogd
     '-update
```

Se puede observar a través del gráfico los cinco procesos bajo el shell del usuario, así como bajo que proceso se ejecutó el shell, y también que proceso es padre y que proceso es hijo.

El segundo comando relacionado con *ps* es *top*. Éste comando, no sólo muestra los procesos actuales, sino que automáticamente se va actualizando para mostrar los cambios acontecidos. Adicionalmente, en la parte superior se muestra información sobre el número de días que ha estado la maquina en marcha, el número de usuarios, la memoria, estadísticas de la memoria de intercambio, etc.

Mientras el comando *top* está en marcha, se pueden usar las siguientes teclas para interactuar con él:

h	Ayuda
q	Salir
s	Cambia el tiempo entre actualizaciones (por defecto, 5 segundos)
espacio	Actualizar ahora en lugar de esperar al siguiente intervalo de actualización
u	Muestra un único usuario

Finalizando un proceso

Bajo circunstancias normales, un proceso hijo actúa bajo el padre que lo ha creado. Cuando el proceso hijo ya no es necesario, desaparece. Algunas veces, sin embargo, los procesos se convierten en procesos 'fugitivos', y aunque no sea necesario que se sigan ejecutando, continúan su ejecución consumiendo recursos innecesarios.

Un proceso padre no puede (y no debe) finalizar su ejecución mientras tenga procesos hijos asociados a él que estén en funcionamiento. Teniendo ésto en cuenta, cuando un proceso hijo no puede finalizar correctamente su ejecución, origina que el proceso padre se quede en un estado inconsistente, y que no pueda, a su vez, terminar su ejecución, quedando el proceso padre (y el o los hijos 'colgados') en un estado conocido como 'zombie', haciendo uso de recursos innecesarios del sistema.

Un ejemplo para entender todo esto: el shell de un usuario ejecuta un proceso (A), que no puede hacer todo por si mismo, así que ejecuta otro proceso (B), que a su vez ejecuta otro proceso (C).
Pueden suceder entonces varias cosas:

- Bajo condiciones normales, cuando el proceso C termina su ejecución, se lo notifica al proceso B, y desaparece (C). El proceso B trata la información, notifica los datos al proceso A, y muere (B). El proceso A, hace lo propio con los datos recibidos, y retorna la información al shell del usuario, y entonces muere (A).

- En condiciones anormales, supongamos que el proceso C, después de pasar la información al proceso padre (el proceso B), no muere. Continua ejecutándose, lo que impide que el proceso B finalice, dado que tiene un proceso hijo (C) en marcha. El proceso B trata la información y la reporta hacia el proceso padre (A), que a su vez, devuelve la información hacia el shell que lo originó.
  Tanto el proceso A como el proceso B, no pueden finalizar su ejecución dado que tienen procesos hijos en marcha. Así pues, un error en el proceso C, que hace que se quede en ejecución cuando no debería, origina que haya tres procesos en marcha en el sistema, consumiendo recursos de forma innecesaria.

- Otro tipo de problema, podría darse de la siguiente manera: el proceso C, como antes, entra en un estado inestable, y no finaliza su ejecución. Aun así, el proceso B, acaba su ejecución y desaparece. El proceso A, también finaliza dado que su hijo, el proceso B, ha finalizado. Así pues, se queda únicamente el proceso C en marcha (en estado inestable), pero ahora, no tiene procesos padre a los que reportar.

Para resolver los problemas que pueden ocasionar estos procesos extraños, se puede usar el comando *kill*.

La sintaxis del comando *kill* es la siguiente:

kill {opcion} PID

Así por ejemplo, para acabar con el proceso *cat* la sentencia es:

```
$ ps -f
UID PID PPID C STIME TTY TIME CMD
root 19605 19603 0 Aug10 pts/0 00:00:34 bash
root 30089 19605 0 Aug20 pts/0 00:00:00 vi fileone
root 30425 19605 0 Aug20 pts/0 00:00:00paste -d fileone filetwo?
root 32040 19605 0 Aug22 pts/0 00:00:00 cat
root 1183 19605 0 Aug23 pts/0 00:00:00 awk -F: questions
root 30900 19605 0 14:25 pts/0 00:00:00 ps -f

$ kill 32040
```

Se pide 'amablemente' que el proceso acabe su ejecución. El termino 'amablemente' se usa dado que hay 32 posibles señales que se pueden enviar a un proceso para que acabe su ejecución, y ésta es la manera más elegante y más segura de acabar con un proceso, se pide que finalice de forma ordenada..
En muchas ocasiones, el proceso ignorará la petición de finalizar su ejecución. Cuando ésto suceda, se puede usar cualquiera de las otras 32 señales para acabar con el proceso.

Entre otras, algunas posibilidades son:
- 1 Colgar/desconectar (hangup/disconnect)
- 2 Usando la secuencia de interrupción (Ctrl+C)
- 3 Salir (quit)
- 9 Sin esperar, salir inmediatamente, matar (kill)
- 15 Por defecto, terminar

Para ver la lista de las posibles señales en el sistema, se usará el comando *kill -l*, y para conocer mejor cuales son las acciones que toman cada una de las opciones se hará uso del comando *man*:

```
$man kill
```

Suponiendo que el proceso *cat* no finaliza su ejecución, la secuencia de operaciones será:

```
  $ ps -f
UID PID PPID C STIME TTY TIME CMD
root 19605 19603 0 Aug10 pts/0 00:00:34 bash
root 30425 19605 0 Aug20 pts/0 00:00:00 paste -d fileone filetwo?
root 32040 19605 0 Aug22 pts/0 00:00:00 cat
root 1183 19605 0 Aug23 pts/0 00:00:00 awk -F: questions
root 30996 19605 0 14:25 pts/0 00:00:00 ps -f

$ kill 32040
$ ps -f
UID PID PPID C STIME TTY TIME CMD
root 19605 19603 0 Aug10 pts/0 00:00:34 bash
root 30425 19605 0 Aug20 pts/0 00:00:00 paste-d fileone filetwo?
root 32040 19605 0 Aug22 pts/0 00:00:00 cat
root 1183 19605 0 Aug23 pts/0 00:00:00 awk -F: questions
root 30998 19605 0 14:25 pts/0 00:00:00 ps -f
$ kill -9 32040
[3]- Killed
$ ps -f
UID PID PPID C STIME TTY TIME CMD
root 19605 19603 0 Aug10 pts/0 00:00:34 bash
root 30425 19605 0 Aug20 pts/0 00:00:00 paste-d fileone filetwo?
root 1183 19605 0 Aug23 pts/0 00:00:00 awk -F: questions
root 31000 19605 0 14:25 pts/0 00:00:00 ps -f
```

Se recomienda siempre para finalizar un proceso, intentar primero con la señal 15 (terminar) antes de usar la señal 9 (matar). También se recomienda que se compruebe que un proceso no tenga hijos antes de matarlo. Si existen procesos hijos, primero se deberá finalizar los procesos hijos antes de finalizar el proceso padre.

Segundo plano (background) y primer plano (foreground)

Cuando se ejecuta un proceso, por defecto se ejecuta en primer plano. Cuando un proceso se ejecuta en primer plano, se convierte en el único trabajo en el que puede trabajar el usuario (con el que puede interactuar), la interacción se basa entonces, en que se acabe éste trabajo.

Por ejemplo, cuando un usuario ejecuta el comando *ls -l*, se mostrará por pantalla el resultado, y hasta que no acabe el comando, no se podrá ejecutar ningún otro comando.

Para ejecutar un proceso en segundo plano (background), simplemente se ha de añadir al final del comando el signo ampersand (&). Esta opción permitirá ejecutar más de un comando a la vez:

```
$ sleep 90 &
[5] 31168
```

El número que aparece entre corchetes es igual al número de trabajos que se tienen actualmente ejecutándose en segundo plano. El número que le sigue (en éste caso, el 31168), es el número de proceso de éste trabajo.

El número de proceso del último trabajo puesto en segundo plano, también puede ser referenciado como $!.

Al poner el trabajo a ejecutarse en segundo plano, se permite al usuario poder seguir trabajando e iniciar otros procesos.

Si se quiere esperar a que acabe un proceso para iniciar otro, el comando *wait*, usado junto al numero de proceso (el que se quiere esperar), esperará a que acabe para poder seguir trabajando:

```
$ sleep 120 &
[5] 31175

$ wait 31175
```

El prompt no aparecerá hasta que el proceso 31175 termine su ejecución.

Trabajos en segundo plano

Para ver que trabajos se ejecutan en segundo plano, se usará el comando *jobs*:

```
$ jobs
[1] Stopped vi fileone (wd: ~)
[2]- Stopped paste -d' fileone filetwo ' (wd: ~)
[4]+ Stopped awk -F: questions (wd: ~)
[5] Done sleep 120
```

Los trabajos que han finalizado (3) no aparecerán, y los trabajos que acaban de finalizar (5) aparecerán una única vez (la próxima vez que se ejecute el comando *jobs*, 5 no aparecerá).
El signo más (+) seguido del número de trabajo entre corchetes, indica el trabajo (proceso) más reciente que se puede ejecutar o que se está ejecutando. El siguiente trabajo más reciente se indica con el signo menos (-). La información 'wd: ~' indica el directorio de trabajo del proceso.

La opción **-** añade el número de proceso (PID) a la salida del comando jobs, poniendo **-**p mostrará sólo el número de proceso (PID) de cada proceso y con la opción **-**n mostrará sólo los procesos que están suspendidos.

De ejecución en segundo plano a primero

Se puede mover un trabajo (proceso) que se esté ejecutando en segundo plano (background) a primer plano a través del comando *fg*.

La sintaxis del comando *fg* permite referenciar a un trabajo usando el signo de porcentaje (%) y el número de trabajo.
Por ejemplo, la siguiente secuencia de comandos lanza una espera (*sleep*) de 120 segundos, el comando se lanza en segundo plano, y luego se pasa a primer plano con el comando *fg*:

```
$ sleep 120 &
[5] 31206

$ fg %5
sleep 120
```

Tener en cuenta que el comando que se está ejecutando se muestra en pantalla al pasar a ejecutarse en primer plano.
Al igual que se usa %5, también se puede referenciar a los dos trabajos más recientes utilizando %+ y %- respectivamente.
Si no se sabe el número de trabajo (y no se recuerda el uso del comando jobs), se puede hacer referencia a un trabajo usando una parte de su nombre, usándolo después del signo de porcentaje y el signo de pregunta (?):

```
$ fg %?v
vi fileone
```

De ejecución en primer plano a segundo

El comando opuesto al usado para pasar trabajos a primer plano, es el comando *bg*, que permite mover un trabajo desde primer plano (foreground) a segundo plano (background).

Antes de usar este comando, se deberá suspender la ejecución del trabajo (para volver a obtener el prompt del sistema).
Para suspender la ejecución, se pulsará la secuencia de teclas que corresponde a la señal de suspender un proceso, por defecto CTRL+Z.

Cuando se suspende la ejecución, el trabajo se para y no reanudará su ejecución hasta que se mueva a primer o a segundo plano:

```
$ sleep 180 {presionado de Ctrl+Z }
[5]+ Stopped sleep 180
```

Ejecutando el comando *bg*, se moverá el trabajo a segundo plano, y se cambiará el estado del trabajo a 'en ejecución'.

Manteniendo procesos al cerrar una sesión

Por defecto la mayoría de las shell's al cerrar una sesión envían la señal SIGHUP a los procesos que se ejecutan en segundo plano, provocando que se terminen. Para evitar este comportamiento existe el comando *nohup*, el cual funcionaría del siguiente modo:

```
$ nohup proceso &
```

De esta manera, al cerrar la sesión el proceso se seguiría ejecutando.

Tema 103.6

Modificando la prioridad de los procesos

Introducción

En este capítulo se completa el tema 103.5, se verá como cambiar la prioridad de los procesos, o bien como empezar un proceso con una prioridad determinada.

Los comandos que se verán en este tema son:

- nice
- renice

Así mismo se harán ejercicios sobre los mismos al final del tema, que serán muy parecidos a los realizados en los exámenes.

Éste tema tiene un peso (importancia) de 3 de cara al examen final de la certificación LPI 101. El total de la suma de pesos de todos los temas es de 106.

Modificando la prioridad de los procesos

Modificar la prioridad de ejecución de los procesos. Ejecutar un programa con prioridad más o menos alta, determinar la prioridad de un proceso, cambiar la prioridad de un proceso en ejecución. Incluye el comando nice y los relacionados con él.

El sistema asigna un nivel de prioridad a cada proceso que se está ejecutando en el mismo. El nivel de prioridad por defecto cuando se ejecuta un proceso es 0 (cero). Los números de nivel de prioridad pueden oscilar entre -20 y 19. Cuanto mayor es el número de nivel de prioridad más lentamente se ejecutará el proceso, y por el contrario, cuanto más bajo sea el número, más rápida será la ejecución. Los niveles de prioridad negativos sólo podrán ser asignados por el usuario *root*. Se pueden ver los niveles de prioridad de los procesos con el comando *top* que se estudió anteriormente.

Estableciendo una prioridad concreta a un proceso

Para asignar un determinado nivel de prioridad a un proceso que va a ejecutarse se hará uso del comando *nice* con la siguiente sintaxis:

nice *prioridad comando*

A continuación se muestra un ejemplo de la utilización correcta de este comando, en el que el proceso *netscape* se ejecutará con un nivel de priodidad 10.

```
$ nice 10 netscape
```

Cambiando la prioridad a un proceso

Si se quisiera cambiar la prioridad de un proceso en ejecución o establecer las prioridades de los procesos de un usuario determinado se debería utilizar el comando *renice*. El comando *renice* admite las opciones mostradas en la siguiente tabla:

Opciones del comando *renice*

Opción	***Descripción***
-g	Establece el nivel de prioridad para los procesos ejecutados por miembros del grupo especificado.
-u	Establece el nivel de prioridad para los procesos ejecutados por el usuario especificado.
-p	Establece el nivel de prioridad para el proceso especificado.

Éste es un ejemplo del uso del comando para establecer el nivel de prioridad de un proceso que se está ejecutando actualmente en el sistema. En este caso establecería un nivel de prioridad 10 para el proceso número 408.

```
$ renice 10 -p 408
```

Tema 103.7

Buscando cadenas de texto usando expresiones regulares

Introducción

En éste capítulo se verá como usar expresiones regulares para usar y filtrar ficheros y cadenas de texto. Se crearán construcciones simples con varios elementos de notación así como herramientas para las expresiones regulares que permitirán realizar búsquedas en el sistema de ficheros o en contenidos de ficheros.

Los comandos que se verán en este tema son:

- grep
- regexp (Expresiones regulares)
- sed

En este capítulo no se harán ejercicios, ya que hay suficientes ejemplos para trabajar con el tema y crear o modificar los existentes para hacer pruebas.

Este tema tiene un peso (importancia) de 3 de cara al examen final de la certificación LPI 101. El total de la suma de pesos de todos los temas es de 106.

Utilizando las Expresiones Regulares

Anteriormente se describió el "filename globbing" con comodines, lo que nos permite listar o encontrar ficheros con elementos comunes (p.e., nombres de fichero o extensiones) con una sola expresión. Los "file globs" utilizan caracteres especiales como *, que tienen un significado especial en el contexto de la línea de comandos. La shell bash interpreta varios comodines, los suficientes como para manejar el problema relativamente simple del "filename globbing". Otros problemas no son tan simples, y extender el concepto de "glob" para cualquier forma genérica de texto (ficheros, streams de texto, variables de cadenas de programas, etc.) puede ofrecer un amplio abanico de posibilidades. Esto se consigue con la utilización de las Expresiones Regulares.

Dos utilidades importantes para los exámenes LPIC Nivel 1 que utilizan estas expresiones regulares son *grep* y *sed*. Estas utilidades son prácticas para las búsquedas de texto. Hay muchas otras herramientas que hacen uso de las expresiones regulares incluyendo awk, los lenguajes Perl y Python y otras utilidades pero, de momento, no es necesario preocuparse de ellas para los exámenes LPIC de Nivel 1.

Utilizando grep

Hace mucho tiempo, cuando la idea de las expresiones regulares comenzaba a calar, el editor de líneas ed incluía un comando para visualizar las líneas del fichero en edición que coincidiesen con una expresión regular determinada. El comando era:

g/expresión regular/p

Esto es, "de forma global, imprime la línea actual cuando se encuentre una coincidencia con la expresión regular", o de forma más simple y en inglés, "global Expresión Regular print." Esta función era tan práctica que pronto se convirtió en una utilidad independiente a la que se le llamó, apropiadamente, *grep*. Más adelante se expandió la gramática de las expresiones regulares de grep en un nuevo comando llamado *egrep* (por "extended grep"). Se pueden encontrar ambos comandos en cualquier sistema GNU/Linux actual, existiendo ligeras diferencias en la forma de interpretar las expresiones regulares. Para la preparación del examen 101 se usará *grep* que también puede utilizar las expresiones regulares "extendidas" si se incluye la opción -E.

Sintaxis

grep [opciones] regexp [ficheros]

Descripción

Busca en ficheros o en la entrada estándar líneas que contengan alguna coincidencia con la expresión regular regexp. Por defecto, solo se listarán las líneas coincidentes. Cuando se especifiquen varios ficheros en la búsqueda, grep incluirá el nombre del fichero como prefijo en las líneas listadas.
En la tabla 7-1 se pueden observar algunas opciones del comando.

Tabla 7-1 Opciones utilizadas frecuentemente

Opción	***Uso***
-c	Muestra solo un contador de las líneas coincidentes pero no las líneas en si mismas.
-h	Muestra las líneas coincidentes pero no los nombres de ficheros en búsquedas de múltiples ficheros.
-i	Ignora las mayúsculas y minúsculas de tal forma que abc coincidiría tanto con abc como con ABC.
-n	Muestra las líneas coincidentes precedidas por su número de línea. Cuando se utiliza con varios ficheros se incluye tanto el nombre del fichero como el número de línea.
-v	Imprime todas las líneas que no coinciden con *regexp*. Esta es una opción útil e importante, a veces se quiere utilizar expresiones regulares no solo para seleccionar información, sino para eliminar información. La utilización de -v invierte la salida de ésta forma.

Ejemplos:
Como las expresiones regulares pueden contener tanto metacaracteres como literales, podemos utilizar grep con una expresión regular totalmente literal.
Por ejemplo, para encontrar todas las líneas en fichero1 que contengan tanto “Linux” como “linux” se podría utilizar *grep* de la siguiente manera:

```
$ grep -i linux fichero1
```

En este ejemplo, la expresión regular es, simplemente, "linux.". La L mayúscula en "Linux" es coincidente gracias a la opción -i (ignorar mayúsculas/minúsculas). Ésto está bien para expresiones literales comunes. No obstante, en situaciones en las cuales regexp incluya metacaracteres que también sean caracteres especiales de la shell (como $ o *), regexp debe estar entrecomillada para evitar la expansión de la shell y poder pasar estos metacaracteres a grep.

Como un ejemplo simplista de ésto último, suponiendo que se tiene en el directorio local unos ficheros llamados abc, abc1, y abc2. Cuando se usa el comando *echo* de bash con la expresión comodín abc* se listarán todos los ficheros que comiencen por abc, como se muestra a continuación:

```
$ echo abc*
```

Ahora suponiendo que estos ficheros contienen líneas con las cadenas abc, abcc, abccc y similares y que se quiere emplear grep para encontrarlas. Se puede utilizar la expresión comodín abc* para que se expanda para todos los ficheros abc como se mostró con echo más arriba, y también se podría utilizar una expresión regular idéntica abc* para encontrar todas las líneas que contengan abc, abcc, abccc, etc. Si no se utiliza las comillas para evitar la expansión de la shell, el comando sería:

```
$ grep abc* abc*
```

Después de la expansión de la shell quedaría:

```
$ grep abc abc1 abc2 abc abc1 abc2  # ¡¡incorrecto!
```

¡Esto no es lo que se pretendía! grep buscará solo la expresión literal *abc,* porque es lo que

encuentra como primer argumento. Para evitar ésto, se encierra la expresión regular con comillas simples o dobles para protegerla de la expansión de la shell:

```
$ grep 'abc*' abc*
```

o:

```
$ grep "abc*" abc*
```

Después de la expansión, ambos ejemplos obtienen los mismos resultados:

```
$ grep abc* abc abc1 abc2
```

Ésto era lo que se pretendía. La expresión regular abc* se buscará en los tres ficheros abc, abc1, y abc2. Es una buena práctica habituarse a entrecomillar las expresiones regulares en la línea de comandos para evitar problemas similares, éstos no son, ni mucho menos, obvios ya que la expansión de la shell es invisible a no ser que se utilice el comando *echo*.

En el examen: Es usual la aparición de preguntas sobre la utilización de grep y sus opciones. Se debería estar familiarizado con lo que hace cada opción así como con el concepto de utilizar la salida de otros comandos como entrada de grep mediante pipes (|) para buscar coincidencias.

Utilizando sed

Anteriormente se presentó *sed*, el editor de streams, se comentó como *sed* utiliza direcciones para localizar el texto sobre el que operar. Entre los mecanismos de direccionamiento mencionados estaba la utilización de expresiones regulares delimitadas entre barras inclinadas.
La sintaxis que se vió de *sed* era:

```
sed [opciones] 'comando1' [ficheros]
sed [opciones] -e 'comando1' [-e 'comando2'] [ficheros]
sed [opciones] -f script [ficheros]
```

Descripción

Se observa que comando1 está escrito entre comillas simples, esto se ha hecho por las mismas razones que se hizo anteriormente con grep, el texto en comando1 debe ser protegido de la posible expansión y evaluación de la shell.
La parte de dirección de un comando sed puede contener expresiones regulares encerradas entre barras inclinadas. Por ejemplo, para mostrar el contenido de fichero1 exceptuando las líneas en blanco se podría utilizar la opción eliminar de sed (d) de la siguiente forma:

```
$ sed '/^$/ d' fichero1
```

En este caso, la expresión regular ^$ (^ indica principio de línea, $ indica final de línea; así por tanto, ^$ busca una línea en la que el principio y el final coinciden, es decir, una línea en blanco) coincide con las líneas en blanco y la opción d elimina esas líneas coincidentes de la salida de *sed*.

Entrecomillado

Como se mostró en los ejemplos de *grep* y *sed* es necesario entrecomillar los metacarácteres de las expresiones regulares si se quiere preservar sus funciones especiales. Si no se hace ésto, se pueden producir resultados inesperados cuando la shell interprete los metacaracteres como caracteres de file globbing. Hay tres maneras de entrecomillado que pueden utilizarse para preservar los caracteres especiales:

- \ (una barra invertida sin comillas)
  Si se pone una barra invertida antes de un carácter especial, éste no será interpretado por la shell, sino que pasará sin alteración hacia el comando que se está introduciendo.
  Por ejemplo, el metacaracter * podría utilizarse en una expresión regular de esta forma:

```
$ grep abc\* abc abc1 abc2
```

  Aquí se buscará en los ficheros *abc*, *abc1*, y *abc2* la expresión regular *abc**.

- *Comillas simples*
  Encerrar los metacarácteres con comillas simples también los protege de la interpretación de la shell. Se asume el valor literal de todos los caracteres entre comillas simples.

- *Comillas dobles*
  Encerrar los metacarácteres con comillas dobles tiene el mismo efecto que con comillas simples con la excepción de los caracteres $, ' (comilla simple), y \ (barra invertida). Tanto $ como ' mantienen sus significados especiales dentro de las comillas dobles. La barra invertida mantiene su significado especial cuando va seguida de $, ', otra barra invertida, o un carácter de nueva línea.

En general, las comillas simples son más seguras a la hora de preservar las expresiones regulares.

En el examen: se ha de prestar atención especial a los métodos de entrecomillado utilizados para preservar los carácteres especiales porque las distintas formas de entrecomillar no tienen por que producir los mismos resultados.

Expresiones Regulares

GNU/Linux ofrece muchas herramientas de procesamiento de texto para los administradores de sistemas. Muchas, como *sed* y los lenguajes *awk* y *Perl*, son capaces de editar automáticamente varios ficheros proporcionando una amplia gama de posibilidades para el procesamiento de textos. Para aprovechar estas prestaciones se necesita ser capaz de definir y delimitar segmentos de texto específicos dentro de ficheros, desde streams de texto y variables de cadena. Una vez identificado el texto que se busca se puede utilizar una de esas herramientas o lenguajes para hacer lo que se necesite con el texto seleccionado.

En conjunto, a éste lenguaje y a las secuencias en sí mismas, se les llama expresiones regulares (a menudo se abrevia como regexp o regex). Aunque las expresiones regulares son conceptualmente similares a los file globs, existen muchos más carácteres especiales para las expresiones regulares extendiendo la utilidad y las prestaciones de las herramientas que las interpretan.

Hay libros completos sobre las expresiones regulares (como el excelente y muy legible

Mastering Regular Expresions de Jeffrey E. F. Friedl (ISBN: 1-56592-257-3), publicado por O'Reilly & Associates). El examen 101 requiere del uso de expresiones regulares y de las herramientas relacionadas específicamente para realizar búsquedas desde fuentes de texto. Esta sección cubre solo lo básico de las expresiones regulares, sería muy recomendable profundizar en este tema cuando se tenga ocasión.

Sintaxis de las Expresiones Regulares

Sería razonable asumir que hay alguna especificación que define como se construyen las expresiones regulares. Desafortunadamente no hay ninguna. Las expresiones regulares han sido tradicionalmente incorporadas como una prestación en diferentes herramientas con diferentes grados de consistencia e integridad. El resultado ha sido un caso similar al del carro delante del caballo, en el que las utilidades y lenguajes han ido definiendo su propio tipo de sintaxis para las expresiones regulares, cada uno con sus propias extensiones e idiosincrasias. La definición formal de la sintaxis de las expresiones regulares vino después, así como los esfuerzos para hacerla más consistente. Las expresiones regulares están formadas por cadenas estructuradas de texto o secuencias. Estas secuencias están compuestas por dos tipos de caracteres:

Metacarácteres

Al igual que los caracteres de file globbing, los metacarácteres de las expresiones regulares toman un significado especial en el contexto de la herramienta en la que están siendo utilizados. Hay unos pocos metacaracteres que generalmente se consideran como parte del "conjunto extendido" de metacarácteres, específicamente aquellos introducidos en *egrep* después de la creación de *grep*. Ahora la mayoría de esos pueden ser manejados por *grep* utilizando la opción -E.

Ejemplos de metacaracteres serían el símbolo ^, que significa "el principio de una línea" y el signo $, que significa "el final de una línea". En las tablas 7-2, 7-3 y 7-4 se puede ver una lista completa de metacaracteres.

Literales

Todo lo que no es un metacarácter es, simplemente, texto plano o texto literal.
A menudo es de ayuda considerar las expresiones regulares como un lenguaje en sí mismo donde el texto literal actuaría como palabras y frases. La "gramática" del lenguaje estaría definida mediante el uso de los metacarácteres. Los dos se combinarían de acuerdo a unas reglas específicas (las cuales, como se mencionó antes, podrían diferir ligeramente entre las diferentes herramientas) para comunicar ideas y hacer algún trabajo real. Cuando se construyen expresiones regulares se utilizan metacarácteres y literales para especificar tres ideas básicas sobre el texto de entrada:

- Referencia de posición

Se utiliza una referencia de posición para especificar la posición de uno o más conjuntos de caracteres en relación a la línea de texto completa (como, por ejemplo, el principio de una línea).

- Conjuntos de caracteres

Un conjunto de caracteres busca coincidencias en el texto. Podría ser una serie de literales, metacaracteres que coincidan con uno o varios caracteres, o combinaciones de ambos.

- Modificadores cuantitativos

Los modificadores cuantitativos siguen a un conjunto de caracteres e indican el número de veces que el conjunto ha de ser repetido. Estos caracteres "dan elasticidad" a una expresión regular

permitiendo que las coincidencias tengan longitud variable.
La próxima sección lista los metacaracteres más comunes. Los ejemplos dados con los metacaracteres son muy simples y solo pretenden demostrar el uso de ese metacaracter en cuestión. Más adelante se verán expresiones regulares más complejas.

Ejemplos de Expresiones Regulares

Una vez dejados los detalles desagradables, se ponen ejemplos de uso prácticos de expresiones regulares que podrían resultar útiles.

Anclajes

Las referencias de posición se utilizan para describir información sobre posiciones. La tabla 7-2 lista los caracteres de anclaje.

Tabla 7-2. Anclajes de Posición de Expresiones Regulares

Expresión Regular	***Descripción***
^	Coincide con el principio de línea. Esta interpretación sólo tiene sentido cuando el carácter ^ está en el lado izquierdo de la expresión regular.
$	Coincide con el fin de línea. Esta interpretación sólo tiene sentido cuando el carácter $ está en el lado derecho de la expresión regular.

Ejemplo 1
Muestra todas las líneas de fichero1 en las cuales aparezca la cadena "Linux" en el principio de la línea, o sea, que comiencen por dicha cadena:

```
$ grep '^Linux' fichero1
```

Ejemplo 2
Muestra las líneas en fichero1 donde el último carácter es una "x":

```
$ grep 'x$' fichero1
```

Muestra el número de líneas vacías en fichero1 encontrando líneas con nada entre el principio y el final:

```
$ grep -c '^$' fichero1
```

Muestra todas las líneas de fichero1 que contengan sólo la palabra "nulo":

```
$ grep '^nulo$' fichero1
```

Grupos y rangos

Pueden situarse caracteres en grupos y rangos para hacer expresiones regulares más eficientes, como se muestra en la tabla 7-3.

Tabla 7-3. Conjuntos de caracteres de las expresiones regulares

Expresión Regular	***Descripción***
[abc] [a-z]	Grupos y rangos de caracteres independientes. En la primera forma se hace coincidir cualquier carácter independiente de entre los caracteres `a`, `b`, o `c`. En la segunda forma, se hace coincidir cualquier carácter independiente de entre el rango acotado por los caracteres `a` y `z`. Los corchetes son sólo para agrupar y no hacen coincidencia por si mismos.
[^abc] [^a-z]	Coincidencia inversa. Se hace coincidir cualquier carácter independiente que no esté entre `a`, `b`, o `c` o en el rango `a-z`. Se ha de tener cuidado con confundir esta inversión con el carácter de anclaje `^` que se describió anteriormente.
\<palabra\>	Coincidencia de palabras. Las palabras son esencialmente conjuntos de caracteres rodeados de espacios en blanco o situados junto al inicio o fin de la línea o un signo de puntuación. Las barras invertidas son imprescindibles para que se interpreten de esta forma < y >.
. (un punto)	Coincide con cualquier carácter (sólo uno), excepto el de fin de línea.
\	Como se mencionó en la sección de entrecomillado, desconecta (escape) el significado especial de carácter que le sigue convirtiendo metacaracteres en literales.

Ejemplo 1
Muestra todas las líneas de *fichero1* que contengan "Linux", "linux", "TurboLinux" y similares:

```
$ grep '[Ll]inux' fichero1
```

Ejemplo 2
Muestra todas las líneas de *fichero1* que contengan tres dígitos numéricos consecutivos:

```
$ grep '[0-9][0-9][0-9]' fichero1
```

Ejemplo 3
Muestra todas las líneas de *fichero1* que comiencen por cualquier carácter que no sea un dígito numérico:

```
$ grep '^[^0-9]' fichero1
```

Ejemplo 4
Muestra todas las líneas de *fichero1* que contengan la palabra completa "Linux" o "linux" pero no "LinuxOS" o "TurboLinux":

```
$ grep '\<[Ll]inux\>' fichero1
```

Ejemplo 5
Muestra todas las líneas de fichero1 con cinco o más caracteres en una línea (excluyendo el carácter de nueva línea):

```
$ grep '.....' fichero1
```

Ejemplo 6
Muestra todas las líneas no vacías de fichero1 (aquellas que tienen al menos un carácter):

```
$ grep '.' fichero1
```

Ejemplo 7
Muestra todas las líneas de *fichero1* que contienen un punto (como . es un metacarácter, se ha de usar el codigo de escape):

```
$ grep '\.' fichero1
```

Tabla 7-4. Modificadores de Expresiones Regulares

Expresión Regular	***Descripción***
*	Coincide con un número indeterminado (cero o más) del carácter (o expresión regular) que lo precede.
?	Coincide con cero o una instancia de la expresión regular que lo precede. Éste modificador es una función "extendida" y solo se dispone en *grep* cuando se utiliza la opción -E.
+	Coincide con una o más instancias de la expresión regular que lo precede. Este modificador es una función "extendida" y solo se dispone en *grep* cuando se utiliza la opción -E.
\{n,m\}	Coincide con un rango de ocurrencias del carácter o la expresión regular que precede ésta construcción. \{n\} coincide con n ocurrencias, \{n,\} coincide con al menos n ocurrencias, y \{n,m\} coincide con cualquier número de ocurrencias entre n y m inclusive. Las barras invertidas son imprescindibles y permiten la interpretación de { y }.
|	Alternancia. Coincide tanto con la expresión regular especificada antes o después de la barra vertical. Este modificador es una función "extendida" y solo se dispone en *grep* cuando se utiliza la opción -E.

Ejemplo 1
Muestra todas las líneas de *fichero1* que contengan "ab", "abc", "abcc", "abccc", y similares:

```
$ grep 'abc*' fichero1
```

Ejemplo 2
Muestra todas las líneas de *fichero1* que contengan "abc", "abcc", "abccc", y similares, pero no "ab":

```
$ grep 'abcc*' fichero1
```

Ejemplo 3
Muestra todas las líneas de fichero1 que contengan dos o más dígitos numéricos consecutivos:

```
$ grep '[0-9][0-9][0-9]*' fichero1
```

Ejemplo 4
Muestra líneas de fichero1 que contengan "fichero" (porque ? puede hacer coincidir cero ocurrencias), “fichero1”, o “fichero2”:

```
$ grep -E 'fichero[12]?' fichero1
```

Ejemplo 5
Muestra todas las líneas de fichero1 que contengan al menos un dígito numérico:

```
$ grep -E '[0-9]+' fichero1
```

Ejemplo 6
Muestra todas las líneas de fichero1 que contengan "111," "1111," o "11111" en una línea y nada más:

```
$ grep '^1\{3,5\}$' fichero1
```

Ejemplo 7
Muestra todas las líneas de fichero1 que contengan cualquier número de tres, cuatro o cinco dígitos:

```
$ grep '\<[0-9]\{3,5\}\>' fichero1
```

Ejemplo 8
Muestra todas las líneas de fichero1 que contengan "Happy", "happy", "Sad", "sad", "Angry", o "angry":

```
$ grep -E '[Hh]appy|[Ss]ad|[Aa]ngry' fichero1
```

Secuencias básicas de las Expresiones Regulares

Ejemplo 1
Coincide con cualquier letra:

```
[A-Za-z]
```

Ejemplo 2
Coincide con cualquier signo (no letra o número):

```
[^0-9A-Za-z]
```

Ejemplo 3
Coincide con una letra mayúscula seguida de cero o más letras minúsculas:

```
[A-Z][a-z]*
```

Ejemplo 4
Coincide con un número de la Seguridad Social de USA (123-45-6789) especificando grupos de tres, dos y cuatro dígitos separados por guiones:

```
[0-9]\{3\}-[0-9]\{2\}-[0-9]\{4\}
```

Ejemplo 5
Coincide con un importe en dólares utilizando un signo de dólar escapado, cero o más espacios o dígitos numéricos, un punto escapado y dos dígitos más:

```
\$[ 0-9]*\.[0-9]\{2\}
```

Ejemplo 6
Coincide con el mes de mayo y su abreviatura, "may". La interrogación marca cero o una instancia del carácter *o* :

```
mayo?
```

Utilizando Expresiones Regulares como direcciones en *sed*

Estos ejemplos son comandos que se proporcionarían a sed. Por ejemplo, los comandos podrían situarse en lugar de comando1 en esta norma de uso:

$ sed [opciones] 'comando1' [ficheros]

Comandos que podrían aparecer en un script independiente de *sed.*

Ejemplo 1
Elimina líneas en blanco:

```
/^$/d
```

Ejemplo 2
Elimina cualquier línea que no contenga #noborrar:

```
/#noborrar/!d
```

Ejemplo 3
Elimina líneas que contengan sólo espacios en blanco (espacios o tabulaciones). En éste ejemplo tab significa el carácter tab y va precedido por un espacio:

```
/^[ tab]*$/d
```

Ejemplo 4
Elimina líneas que comiencen por puntos o signos #:

```
/^[.#]/d
```

Ejemplo 5
Sustituye cualquier número de espacios por un sólo espacio donde quiera que aparezca en la línea:

```
s/  */ /g
```

Ejemplo 6
Sustituye abc por def de la línea 11 a la 20, donde quiera que aparezca en la línea:

```
11,20s/abc/@@@/g
```

Ejemplo 7
Cambia los caracteres a, b, y c por el carácter @ desde la línea 11 a la 20, donde quiera que aparezca en la línea:

```
11,20y/abc/@@@/
```

En el examen
Asegurarse de que se tienen claras la diferencia entre file globbing y la utilización de expresiones regulares.

Tema 103.8

Usando el editor vi

Introducción

En este capítulo se verá el funcionamiento básico de uno de los editores de texto plano más famosos en el mundillo GNU/Linux. Navegar, insertar, copiar, mover, pegar y buscar, serán algunas de las tareas que se verán en este capítulo, y que servirán de ahora en adelante para editar los archivos de texto, ya sean textos o bien ficheros de configuración del sistema.

El único comando que se verá será el vi y algunas de sus funciones:

/, ?
h, j, k , l
G, H, L
i, c, d, dd, p, o, a
ZZ, :w!, :q!, :e!

Este tema tiene un peso (importancia) de 1 de cara al examen final de la certificación LPI 101. El total de la suma de pesos de todos los temas es de 106. A pesar de ello se recomienda el conocimiento de este potente y rápido editor.

Abriendo ficheros para su edición

Para abrir un fichero y así poder editarlo con vi, se usa el comando siguiente:

```
$vi  nombrefichero
```

Si no se especifica un nombre de fichero, vi abre uno en blanco, y luego se puede especificar el nombre al guardarlo más tarde. Hay que tener en cuenta los permisos de lectura/escritura del fichero, o no se podrán guardar los cambios.

Si se sabe la línea que se necesita editar, se puede ir directamente a esa línea usando la siguiente sintaxis:

```
$vi +<num_línea> nombrefichero
```

Con esto se le indica a vi que empiece editando el fichero en la línea dada. Otra opción para saltar rápidamente a un punto del fichero es especificar una cadena de búsqueda por donde empezar. Sería con la siguiente sintaxis:

```
$vi +/telnet inetd.conf
```

Esto arrancaría vi, abriría el fichero inetd.conf, y luego iría a la primera palabra que coincida con telnet.

vi tiene tres modos de actuar:

- Modo comando
- Modo insertar
- Modo última línea

Cuando arranca vi, el modo por defecto es por comando. En modo comando se puede mover el cursor por todo el documento, poner comandos, borrar y añadir texto, o cambiar a modo insertar. Para volver al modo comando hay que pulsar la tecla Esc. Cuando se está en modo insertar, sólo se puede insertar texto y usar la tecla suprimir para borrar los errores. No se permite en este modo el mover el cursor por el documento. Para ello hay que volver al modo comando. El modo última línea se usa para entrar comandos complejos. Para entrar en ese modo se escriben desde modo comando dos puntos (:). Después de completar el comando se vuelve al modo comando.

Recordar que en vi los comandos son sensibles a las mayúsculas/minúsculas. J es diferente de j.

Saliendo de vi y guardando los archivos

Para salir de vi se hace desde el modo última línea, de manera que el comando debe empezar con dos puntos desde el modo comando.

Guardar sin salir.
Para guardar el fichero, se usa el comando :w- Para evitar mayores problemas se debería hacer esto con frecuencia.

Guardar y salir.
Para salir de vi guardando los cambios, se puede usar tanto el comando :wq como simplemente ZZ-

Salir sin guardar.
Si se necesita rehacer los cambios que se han hecho a un fichero, se debe salir de vi sin guardar los cambios. Este se puede hacer con el comando :q! La exclamación anula la necesidad de guardar el fichero antes de poder salir del programa.

Moviendo el cursor

Vi es un editor en modo texto, por lo que no está disponible el soporte del ratón. Esto suele a veces desalentar a los nuevos usuarios, superándose rápidamente con un poco de práctica. Vi ofrece numerosos atajos para moverse rápidamente por los ficheros, de manera que los usuarios no se atranquen con los simples movimientos del cursor.

La mayoría de las veces se pueden usar las teclas de flechas para moverse por el documento. Pero no todos los sistemas tienen estas teclas, y a veces las incompatibilidades de la emulación de un terminal causa que no funcionen como se espera. Las teclas para mover el cursor se muestran a continuación, colocadas según su función:

```
        k
   h         l
        j
```

La tecla - (menos) se puede sustituir por k, y la tecla + (más) por la j. Se deben aprender estas teclas incluso aunque se use siempre las flechas. Muchas otras aplicaciones usan las teclas de movimiento del cursor de vi dado que son populares y familiares. Por si solas, mueven el cursor una línea o carácter. Para mover más, se puede usar un número, como 4j para moverse 4 líneas hacia abajo.

Para moverse por el fichero más rápidamente, se puede usar los comandos de página completa o media página. Estos son Ctrl+f y Ctrl+b para moverse hacia delante o hacia detrás una página, y Ctrl+d y Ctrl+u para moverse hacia delante o hacia atrás media página.

También se pueden usar los números de línea para moverse por un fichero. Para ver los números de las líneas se usa el comando Ctrl+g.. Para moverse a una línea específica se usa el comando nG, donde n es el número de línea.

Hay muchos otros comandos para mover el cursor. La mayoría de ellos se muestran en la siguiente tabla. Hay que recordar que se puede hacer que un comando se repita poniéndole un número delante.

Tabla 8-1 Comandos para mover el cursor

Tecla	Función
h	Mueve el cursor un carácter hacia la izquierda
j	Mueve el cursor hacia abajo una línea
k	Mueve el cursor hacia arriba una línea
l	Mueve el cursor un carácter hacia la derecha
Ctrl-G	Muestra el número de la línea en la que se encuentra
nG	Va a la línea especificada en n
Ctrl-f	Avanza una pantalla
Ctrl-b	Retrocede una pantalla
Ctrl-d	Avanza media pantalla
Ctrl-u	Retrocede media pantalla
Ctrl-e	Se mueve una línea hacia arriba
Ctrl-y	Se mueve una línea hacia abajo
w	Va al principio de la siguiente palabra
e	Va al final de la siguiente palabra
E	Va al final de la siguiente palabra. A diferencia del comando e, E considera la puntuación como parte de la palabra.
b	Va al principio de la palabra anterior.
0	Va al principio de la línea
^	Va a la primera palabra de la línea en la que se encuentra.
$	Va al final de la línea
Enter	Va a la siguiente línea.
-	Va al principio de la línea anterior.
G	Va al final del fichero.
%	Va a la agrupación que coincida
H	Va a la primera línea de la pantalla.
M	Va a la mitad de la pantalla.
L	Va al final de la pantalla.
n\	Mueve el cursor a la columna n.

Guardando una parte del fichero

No solo se puede salvar el fichero entero, vi permite guardar parte de un fichero especificando el número de líneas que se quiere escribir. Se puede uno figurar que cada línea se puede referenciar por un número, y esta operación se haría con la siguiente sintaxis:

: primera_línea , ultima_línea w nombreFichero

Especial atención a la w que va detrás de la última línea, que se usa para identificar la operación como de escritura. Hay dos caracteres que se pueden usar para especificar el número de línea:

$ significa la última línea del fichero
. significa la línea actual

Tabla 8-2 Algunos ejemplos de comandos para guardar parte de un fichero:

Secuencia	Resultado
:., 12w ficheroNuevo	Guarda la línea desde donde está el cursor hasta la línea 12 en un fichero llamado ficheroNuevo
:2, 5w ficheroNuevo	Guarda las líneas 2 a 5 en un fichero llamado ficheroNuevo.
:12, $w ficheroNuevo	Guarda las líneas desde la 12 hasta el final del fichero en un fichero llamado ficheroNuevo

Añadiendo texto

Se puede añadir texto de diferentes maneras en vi. Cualquiera de ellas precisa que el usuario esté en modo comando.

Insertando texto

Para entrar en modo insertar, usar el comando i. Esto hace que el texto se inserte en el documento dondequiera que esté el cursor. Para volver al modo comando, usar la tecla Esc.

Añadiendo texto
Para añadir texto desde la localización del cursor, usar el comando a. Para añadirlo al final de la línea actual, usar el comando A.

Creando una línea nueva
Para empezar una nueva línea de texto se pueden usar dos comandos. El primero, o, abre una nueva línea de texto por debajo del cursor. El segundo, O, crea una nueva línea de texto por encima de la localización del cursor.

Cambiando texto

Vi usa un comando doble para cambiar el texto. Cuando se usa, vi marca las secciones a ser cambiadas. Después de marcarlo, el usuario entra el nuevo texto para reemplazar el viejo.

Comienza por el comando c. La segunda parte del comando le dice a vi que texto se ha de cambiar. Para cambiar una palabra, se usa cw, y para cambiar una frase entera cs. Vi usa un espacio para final de palabra y un punto para final de frase. Para cambiar el texto desde la posición actual al final de la línea, usar c$ o C.

Como muchos otros comandos, se puede usar la tecla para repetirlo. Por ejemplo, para cambiar tres palabras se pondría 3cw.

Sustituyendo texto

Para sustituir el texto existente no siempre necesitas contar el número de palabras o frases a cambiar. El comando r permite reemplazar un simple carácter en la posición que esté el cursor, y el comando R permite remplazar todo el texto hasta que se vuelve al modo comando con Esc.

Tabla 8-3 Lista completa de comandos para añadir texto

Comando	Función
i	Inserta texto a la izquierda del cursor.
I	Inserta texto antes del primer caracter en la línea que no es un espacio.
a	Añade texto a la derecha del cursor.
A	Añade texto al final de la línea en la que está.
o	Empieza una nueva línea de texto por debajo de la línea actual.
O	Empieza una nueva línea de texto por encima de la línea actual.
cw	Cambia una palabra.
cs	Cambia la frase actual.
C$ o C	Cambia la línea actual.
r	Reemplaza un solo carácter.
R	Reemplaza texto hasta que se pulsa la tecla Esc.
s	Sustituye texto por el carácter actual.

Borrando texto

Como se puede observar, vi ofrece muchas maneras diferentes de hacer las cosas, y borrar textos no es una excepción.

Borrando caracteres

Para borrar un carácter de texto se usa los comandos x y X. El comando x borra el carácter seleccionado por el cursor, mientras que el comando X borra el carácter que precede al cursor.

La función repetir también se usa con estos comandos. Para borrar ocho caracteres antes del cursor, se usuaria el comando 8X.

Borrando palabras

El comando dw borra la palabra actualmente seleccionada. Para seleccionar una palabra basta con situar el cursor en la primera letra. Para borrar varias palabras a la derecha del cursor poner un número antes o después de la d, como en 3dw o d3w.

Borrando líneas

Para borrar la línea de texto en la que está situado el cursor, usar el comando dd. Para borrar múltiples líneas se añade un número, como por ejemplo 3dd para borrar tres líneas de texto a partir de donde esta el cursor.

Borrando texto desde el cursor.

Dos comandos se usan para borrar todo el texto antes o después del cursor. Para borrar todo el texto

después del cursor hasta el final de línea, usar el comando D, y para borrar todo el texto desde el principio de la línea hasta el cursor, usar d^. Para borrar todo el texto desde la posición actual del cursor hasta el final de la pantalla, usar dL, y para borrar todo el texto hasta el final del fichero, usar dG. Para borrar todo el texto desde el cursor hasta el principio del fichero, usar dlG. Para borrar todo el texto desde el cursor a una línea específica, usar el comando d#$, donde # es el número de la línea.

Tabla 8-4 resumen de los diversos comandos para borrar

Comando	Función
x	Borra el carácter donde se encuentra el cursor.
X	Borra el carácter anterior a donde se encuentra el cursor.
dw	Borra una palabra.
dd	Borra una línea.
D	Borra todo el texto después del cursor hasta el final de la línea.
dL	Borra todo el texto después del cursor hasta el final de la pantalla.
dG	Borra todo el texto desde el cursor hasta el final del fichero.
d^	Borra todo el texto desde el principio de la línea hasta el cursor.
dlG	Borra todo el texto desde el principio del fichero hasta el cursor.
d#$	Borra todo el texto desde el cursor hasta el número de línea especificado.
d$	Borra desde la situación actual del cursor hasta el final de línea
d)	Borra el resto de la frase.
d}	Borra el resto del párrafo.
d0	Borra desde la situación actual del cursor hasta el principio de la línea.
db	Borra la palabra anterior.
dl	Borra una letra.
7dl	Borra siete letras.
7dw	Borra siete palabras.

Copiando y pegando

Como otros editores de texto modernos, vi ofrece las opciones de copiar y pegar texto. Esto puede ser de gran ayuda para realizar con más rapidez trabajos repetitivos, editando textos como por ejemplo shell scripts.

Cortar (yank) y pegar (paste) texto
Para copiar una línea de texto es necesario usar la combinación de teclas yy o Y, añadir un número al inicio de las combinaciones hará que se copie ese número de líneas, por ejemplo 5Y para copiar 5 líneas. Para pegar texto, se usa el comando p.

En vi se usan dos tipos de buffer:

- El predeterminado (unnamed buffer), cuando se copia una porción de texto, esta va a parar a este buffer, sobreescribiéndose el anterior texto que hubiera en él.
- Y el named buffers, estos al contrario que el predeterminado se les llama por un nombre (una letra) y permiten más operaciones como veremos.

Usando yy o Y el texto se copia al buffer (unnamed buffer). Vi soporta hasta 26 buffers diferentes (named buffers). El unnamed buffer es útil para copias rápidas, pero se pueden perder por accidente, por eso es útil tener varios named buffers disponibles (estos no pierden su contenido si no se le indica). Para diferenciar entre un comando y un named buffer se usa el símbolo “ (comillas). Los named buffers son representados por un solo carácter. Cuando se copia texto, las letras minúsculas se usa para sustituir el texto que está en el buffer, sin embargo las mayúsculas se usan para añadir texto al buffer. Por ejemplo, para mover la línea actual al buffer b, se usa:

"bY o "byy

Para copiar 3 líneas al buffer:

"by3

Y para añadir 3 líneas al buffer actual:

"By3

Este es un buen ejemplo de como los comandos pueden ser apilados en vi. Para copiar palabras en vez de líneas se usa el comando yw. Para mover 3 palabras al buffer c, se usaría: "Cy3w

Para pegar texto del buffer c: "Cp

Moviendo texto

Para mover texto de un lugar a otro del documento, se usa el comando dd. Por si mismo, el comando pone el texto en el unnamed buffer, pero también se puede añadir a un named buffer. Para borrar el texto y ponerlo en el buffer se usa el comando: "add

O para añadir 5 líneas: "a5dd

Para pegar el texto movido, se usan los mismos comandos que para pegar texto copiado.

Copiando y moviendo entre ficheros

El unnamed buffer puede ser usado solamente para un único fichero, sin embargo, los named buffers se pueden usar para mover texto entre ficheros. Para realizar esta operación, se pone el texto en un named buffer y se abre otro fichero con el comando :e. Para abrir el fichero llamado lista.txt: :e lista.txt

Una vez el fichero está abierto, puedes pegar el texto como se haría normalmente. Por ejemplo, para mover 5 líneas del fichero original al fichero nuevo:
"a5Y

```
:e lista.txt
"ap
:w
:e lista_vieja.txt
```

Esto copia 5 líneas del fichero lista_vieja.txt, abre el fichero lista_nueva.txt, pega el contenido del buffer, salva el fichero y por ultimo vuelve a abrir el fichero original. La tabla 8-5 hace un resumen de los comandos de copiado y pegado.

Tabla 8-5 Comandos de copiado y pegado

Comando	Uso
yy or Y	Copia una línea en el unnamed buffer
nyy or nY	Copia n líneas al unnamed buffer
yw	Copia una palabra en el unnamed buffer
ynw or nyw	Copia n palabras en el unnamed buffer
y$	Copia el texto desde el cursor hasta final de línea en el unnamed buffer
"ayy	Copia una línea en el named buffer a
"Ayy	Añade una línea en el named buffer a
p	Pega el contenido del buffer a la derecha del cursor
P	Pega el contenido del buffer a la izquierda del cursor
np	Pega n copias del texto a la derecha del cursor
"ap	Pega el contenido del named buffer a a la derecha del cursor
"c3P	Pega 3 copias del buffer c, a la izquierda del cursor
"add	Mueve la línea actual al buffer a
"a5dd	Mueve 5 líneas al buffer a
dw	Borra una palabra y la pone en el unnamed buffer

Buscando texto

Además de las habilidades para manipular texto, vi también ofrece potentes mecanismos para buscar en los ficheros. Vi ofrece la posibilidad no solamente buscar simples palabras o frases como otros editores, sino también de buscar mediante expresiones regulares.

Buscando mediante patrones (pattern matching)

Buscando texto
vi ofrece 2 métodos para buscar en el texto. El primero es muy simple y busca solamente por un carácter. Para buscar la siguiente referencia del carácter a, se pone el comando fa. Para buscar una referencia de a antes del cursor se usa el comando Fa. Si no se quiere tener el cursor situado en el carácter buscado, se pueden usar los comandos T y t. Si se busca por ta, el cursor se situa a la

izquierda de a. Si se usa Ta en el siguiente carácter a la derecha.

Para buscar más de un simple carácter, se usan los comandos / y ?. Cuando se está en el modo comando, se puede teclear el metacaracter / para buscar la primera referencia del texto a la derecha del cursor, para buscar a la izquierda se usa el comando ?, se pueden usar expresiones regulares cuando se está buscando con los metacaracteres / y ?

Repitiendo una búsqueda
Para repetir una búsqueda simple, se usa la tecla ; Para cambiar la dirección de la primera búsqueda simple, se usa el metacaracter , (coma).

Para repetir la búsqueda realizada con / o ?, se pulsa otra vez el mismo metacarácter (/ o ?) seguido de enter. Se puede elegir la misma búsqueda y cambiar de dirección al mismo tiempo. Un acceso directo para repetir la búsqueda en la misma dirección es usando el comando n, y para la dirección opuesta el comando N.

Los caracteres especiales que usa vi con las búsquedas amplían las capacidades del editor de texto más allá de simples palabras o frases. Estos caracteres se les conoce como metacaracteres.

Buscando un carácter simple
Para buscar un carácter simple, se usa el . (punto). Por ejemplo, para buscar casas y casos:
cas.s

Buscando caracteres múltiples.
El * se usa para buscar caracteres múltiples, también buscará el carácter vacío. Por ejemplo, m*n buscará mano, montón, mon, y también mn.

Principio y final de línea .
Se puede indicar que un patrón coincida solamente si está al final o al principio de las líneas. El símbolo ^ define el principio de línea y el símbolo $ define el fin. Recordando que las búsquedas son sensibles a las mayúsculas-minúsculas (case-sensitive). Por ejemplo, el resultado de buscar la cadena “^El” (sin comillas) coincidirá con las dos primeras frases:

El perro se llama Bob.
Elias compró un coche.
el camión en rojo.

El resultado de la cadena “kernel$” coincidirá con la primera y tercera línea :

Ayer puse el nuevo kernel
A Maria no le gusta tocar en el kernel.
Mario tiene problemas con su kernel

Buscando un metacaracter
En algunos casos se necesita buscar un carácter que puede ser interpretado como un metacarater, como un punto o el símbolo del dólar, para evitar la sustitución y que vi interprete exactamente lo que se quiere buscar, se tiene que indicar cuando se trata de una situación como esta (a esto se le llama “escaping the character”) se emplea la barra invertida (\ ) seguido del metacaracter, por ejemplo para buscar “[...] pedro. A la hora [...] Se usa:
pedro\. A la hora

Buscando un rango de caracteres
A veces puede resultar útil la búsqueda de caracteres que se encuentran en un rango dado, esto se consigue poniendo entre corchetes el rango deseado. Por ejemplo para buscar todas las entradas v2.x se usa:
v2\.[1-9]

Recordando que se necesita usar la barra invertida para buscar el . . Todos los números entre el 1 y el 9 se incluirán en el resultado. El carácter ^ se puede emplear para buscar lo contrario de lo que ponemos, por ejemplo para buscar cualquier letra que no sea dos: [^2]

En la tabla 8-6 se da un listado de los metacaracteres más comunes

Tabla 8-6 Metacaracteres más comunes

Metacaracter	función
\	Búsqueda de metacaracteres
.	Cualquier carácter
*	Ninguno o varios caracteres
[]	Rango de caracteres
^	Busca caracteres al principio de línea
$	Busca caracteres al final de línea
[^a]	Excluye letras de la búsqueda
[a-b]	Busca cualquier resultado que esté el rango

Buscando y reemplazando texto

Los mecanismos de vi ofrecen la posibilidad de buscar un texto y reemplazarlo. Estas funciones se pueden usar en determinados bloques de texto a lo largo del fichero. Estos comandos se deben usar desde el prompt ':'. Para buscar y reemplazar el texto en el fichero entero, se utiliza el comando :% y para buscar desde el cursor en adelante el comando :.,$ Para buscar en un determinado rango de líneas se usa :1,5 el cual se aplicará desde la línea 1 hasta la 5. Para las siguientes líneas relativas desde la posición del cursor :/+3, lo cual referencia a la 3ª línea desde la posición actual. La sintaxis para la búsqueda y sustitución es:

:<inicio>,<finalizacion>s/<encontrar>/<reemplazar>

Por ejemplo, para reemplazar la palabra kernel por módulo en la primera instancia de cada línea , se usa

:%s/kernel/modulo

Para hacer lo mismo en todas las líneas se debe añadir la opción g tal como se muestra en el siguiente ejemplo:

:%s/kernel/modulo/g

Esto realiza una búsqueda y sustitución completa automáticamente. Para que el vi actúe de modo interactivo, es decir, que pida confirmación de cada cambio al usuario, se usa la opción c:
:%s/kernel/modulo/gc

La tabla 8-7 muestra los comandos más comunes de las búsquedas

Tabla 8-7 Comandos de búsqueda

Comando	función
fa	Busca la primera referencia de a a la derecha del cursor
Fa	Busca la primera referencia de a a la izquierda del cursor
ta	Se mueve al carácter a la izquierda de a
Ta	Se mueve al carácter a la derecha de a
;	Repite la búsqueda realizada por los comandos f,F,t o T
,	Misma función que ; pero en orden inverso
/string	Busca a la derecha la referencia del string
?string	Busca a la izquierda la referencia del string
n	Repite la búsqueda realizada con / o ?
N	Repite la búsqueda realizada con / o ? en orden inverso
:%	Busca en el fichero entero
:.,$	Busca desde la posición del cursor en adelante
:1,5	Busca en las líneas de la 1 a la 5

Deshaciendo cambios

Por norma general todo el mundo comete errores, y como no podía ser de otro modo, vi ofrece la posibilidad de deshacer los cambios que se han hecho en la línea actual. El comando undo (u) deshace la acción anterior (y sólo la acción anterior). Tiene un complemento -U- el cual deshace todas las acciones previas en la línea en que se encuentre. Es importante observar, sin embargo, que U solo almacena los cambios de una línea (la actual); esto impide que se hagan cambios a cuatro líneas y luego volviendo a la primera se intente deshacer todo. Si realmente se hizo un estropicio se puede volver atrás con la última copia salvada con el comando :e!

Otras operaciones

Con tantas operaciones que se han visto en este capítulo sobre los números de línea – se puede mover a una línea en particular especificando su número, se pueden guardar sólo unas líneas específicas, etc... es a menudo de gran ayuda ver las líneas numeradas. En vi se activa la numeración de líneas cambiando a modo comando con el símbolo de dos puntos (:) - también

llamado modo dos puntos -, mover el cursor al final de la pantalla, entrar el comando set number y presionar Enter. Esto activa la numeración de líneas.

Los números sólo aparecen en el editor, como si los estuviéramos viendo con el comando nl, y no se guardan con el fichero. Set number se puede abreviar como set un (para deshacer el número, usar tanto set nonumber como set nonu). Si sólo se quiere visualizar el número de línea en la que se encuentra actualmente, se puede usar la combinación de teclas Ctrl+G. Los números de línea (así como el número total de líneas dentro del fichero) aparecen al final de la pantalla. Según sea su ejecución, la combinación de teclas puede también mostrar otros items como el porcentaje de fichero antes del cursor, o la columna actual.

Si se desea ejecutar un comando externo dentro del editor, se puede usar la siguiente sintaxis:

:!{comando}

Por ejemplo, para ver una lista de los archivos del directorio actual mientras se trabaja desde vi, el comando sería:

:!ls -l

Este comando muestra la lista y luego indica que se presione Enter para volver al fichero en el que se estaba trabajando.

Si se necesita copiar el contenido de otro fichero dentro del actual, se puede utilizar la siguiente sintaxis:

:r {nombreFichero}

Por ejemplo, para incluir el contenido de un fichero llamado primero dentro del fichero actual, el comando sería:

:r primero

Esto inserta el texto desde el otro fichero directamente en el actual en donde esta situado el cursor.

Ejercicios

Las siguientes preguntas y ejercicios te permitirán repasar los contenidos de este capítulo. Tómate tu tiempo para completarlas y repasa cualquier duda que tengas. El contestar la respuesta correcta no es tan importante como el entender correctamente la pregunta, así que si te es necesario, echa mano de cualquier material que te permita saber exactamente que te están preguntando.

Sentirte cómodo con las preguntas y respuestas de esta sección te ayudara a estar más preparado para el examen de certificación LPI.

Preguntas Pre-Test

1. ¿Qué herramienta se usa para añadir el número de línea a un fichero ?
2. ¿Qué herramienta permite combinar dos ficheros usando campos de unión?
3. ¿Con qué utilidad se crean líneas de una determinada longitud en un fichero?
4. ¿Qué herramienta se usa para ver el contenido de un fichero a la inversa?
5. ¿Qué herramienta permite borrar caracteres de un fichero?
6. ¿Con qué herramienta se puede ver el contenido en hexadecimal de un fichero?
7. ¿Con qué utilidad se pueden ordenar alfabéticamente los contenidos de un fichero?
8. ¿Que comando de búsqueda de ficheros utiliza una base de datos?
9. ¿Donde se encuentra el núcleo (kernel) del sistema?
10. ¿Que tipo de link puede apuntar a distintos sistemas de ficheros?
11. ¿Que comando se utiliza para listar los ficheros de un directorio?
12. ¿Que comando se utiliza para crear un directorio?
13. ¿Que comando puede utilizarse para crear un fichero vacío?
14. ¿Que comando se emplea para buscar un comando en la ruta de PATH?
15. ¿Que comando busca recursivamente un fichero en un directorio?
16. Que fichero contiene la configuración por defecto del shell para cada usuario?
17. Cual es la mejor forma de añadir la variable de entorno PATH para todos los usuarios en un sistema?
18. Como es el tamaño del fichero del historial bash?
19. Que tecla se usa para completar comandos con el bash shell?
20. Que comando permite editar el ultimo comando que introdujiste usando el editor por defecto?
21. Que fichero contiene el historial de comandos?
22. Que comando usuarias para ver la configuración de la variable de entorno HOME?
23. Como ejecutas un programa localizado en el pwd?
24. Que función permite la salida desde un comando para ser usada en lugar del comando?
25. Que opción se usa para operaciones recursivas de un comando?
26. ¿Que caracteres se usan para redirigir la stdout?
27. ¿ Que caracteres se usan para redirigir la stdout y stderr ?
28. ¿ Que utilidad se usa para enviar las salidas de un comando a stdout y a un fichero?

PREGUNTAS TEST

1. ¿Cual de los siguientes ejemplos muestra la utilización de un path absoluto?
   A. ls
   B. ls –al
   C. ls /home/angie
   D. /bin/ls

2. El comando _____ mostrará todas las entradas que comiencen por la letra a en el directorio actual.

3. ¿Cual de los siguientes comandos creará un nuevo fichero con el nombre nuestrogrupo?
   A. file nuestrogrupo
   B. touch nuestrogrupo
   C. ls nuestrogrupo
   D. mkfile nuestrogrupo

4. ¿Que comando se utiliza para cambiar el nombre del fichero nuestrogrupo a migrupo?
   A. rn nuestrogrupo migrupo
   B. rn migrupo nuestrogrupo
   C. mv nuestrogrupo migrupo
   D. mv migrupo nuestrogrupo

5. El comando _____ se utiliza para copiar ficheros con conversión simultánea.

6. ¿Que directorio contiene el núcleo (kernel) del sistema?
   A. /etc
   B. /
   C. /boot
   D. /proc

7. ¿Que directorio contiene los ficheros de configuración del sistema?
   A. /etc
   B. /
   C. /boot
   D. /proc

8. ¿Que directorio contiene el directorio home del usuario root?
   A. /home
   B. /root
   C. /sbin
   D. /usr

9. ¿Que directorio contiene los ficheros de correo?
   A. /proc/mail
   B. /var/spool
   C. /var/mail
   D. /usr/mail

10.¿Que utilidad se utiliza para actualizar la base de datos slocate?
A. locate
B. find
C. whereis
D. updatedb

11. El comando _____ se utiliza para mostrar el path de un comando determinado.

12.¿Cual de los resultados siguientes ocurre cuando se copia un soft link?
A. Se crea una nueva copia del soft link.
B. Se crea un hard link hacia el fichero original.
C. Se crea un hard link hacia el soft link.
D. Se crea una nueva copia del fichero original.

13.¿Qué comando buscará de la linea 2 a la 20 en el archivo records los caracteres 1st y los reemplazara por los caracteres first?
A. sed s2-20/1st/first/ records
B. sed 2-20s/1st/first/ records
C. sed s2,20/1st/first/ records
D. sed 2,20s/1st/first/ records

14.¿Qué comando dividirá el archivo researchpaper en varios archivos que contengan cada uno de ellos 60 líneas? (Escoge todas las respuestas correctas)
A. split -60 researchpaper
B. split -C 60b researchpaper
C. split -C 60 researchpaper
D. split -l 60 researchpaper

15.¿Qué utilidad se usa para combinar las líneas de dos archivos diferentes? (Escoge todas las respuestas correctas)
A. split
B. join
C. paste
D. cut

16.¿Qué usarías para ver las 5 últimas líneas del archivo myfiles?
A. tac myfiles
B. tail myfiles
C. tac -5 myfiles
D. tail -5 myfiles

17.¿Cuál de las siguientes respuestas te permitirían ver el archivo myfiles en formato octal? (Escoge todas las correctas).
A. od myfiles
B. od -t o myfiles
C. od -t x myfiles
D od -o myfiles

18.¿Qué respuesta ordenaría alfabéticamente el fichero mylist, numeraría las líneas y finalmente lo separaría en ficheros que contuvieran 60 líneas cada uno?
A. sort mylist | nl > -60 lists
B. sort mylist > nl > split -60 > lists
C. sort mylist | nl | split -60 lists
D. sort mylist | nl | tee lists | split -60 lists

19.La utilidad _____ sirve para ver el contenido de un fichero de forma inversa. (Rellena)

20.¿Qué utilidad nos muestra el total de líneas de una archivo?
A. nl
B. ln
C. wc
D. tr

21.La utilidad _____ se usa para asegurarse que los archivos se verán igual, sin importar el sistema que se use para visionarlos, convirtiendo las tabulaciones en espacios.

22.¿Qué utilidad intenta convertir las líneas de un archivo en líneas de la misma longitud?
A. nl
B. ln
C. fmt
D. expand

23.¿Con qué utilidad convertiríamos todas las minúsculas de un archivo a mayúsculas?
A. cut
B. sed
C. tac
D. tr

24.¿Cuál de las siguientes respuestas usarías para verificar que las líneas de un archivo están ordenadas alfabéticamente?
A. sort -c
B. sort -d
C. sort -v
D. sort -m

25.Que shell es usada por defecto en los sistemas GNU/Linux?
A. csh
B. rsh
C. bash
D. tcsh

26. Que fichero contiene los shells disponibles para el sistema?
A. /etc/passwd
B. /etc/command
C. /etc/bash
D. /etc/shells

27.Cual es el comando usado para cambiar el shell por defecto para el BASH2?
A. chng -s /bin/bash2
B. chsh -s /bin/bash2
C. shell -c /bin/bash2
D. default -shell /bin/bash2

28.Cual es el formato correcto para introducir un comando en la linea de comandos?
A. command
B. command options
C. command arguments options
D. command options arguments

29.Se pueden poner varios comandos en la misma linea separados porque carácter?
A. .
B. ;
C. ,
D. \

30.Que combinación de teclas es usada para introducir un comando en varias lineas?
A. ""
B. \ Enter
C. / Enter
D. Tab-Enter

31.Que tecla, presionada una vez, es usada para completar los comandos?
A. Tab
B. Esc
C. Enter
D. Ctrl

32.Que tecla, cuando es presionada dos veces, es usada para completar los comandos?
A. Tab
B. Esc
C. Enter
D. Ctrl

33.Que fichero contiene las variables del sistema que tiene el shell bash?
A. /bash
B. /bin/bash
C. /etc/bash
D. /etc/profile

34.. _____hará que se ejecute el script llamado update_mozilla guardado en tu directorio home, desde ese directorio.

35.Cuando se hacen cambios a las variables del entorno, que comando debe de usarse para asegurarse de que los cambios están disponibles para la shell?
A. save
B. remember
C. export
D. echo

36.Que variable de entorno se usa para personalizar el prompt?
A. PS1
B. prompt
C. shell
D. display

37.Que fichero contiene la asignación a la carpeta del usuario home?
A. /etc/home
B. /etc/profile
C. /etc/passwd
D. /etc/users

38.Que comando lista los comandos ejecutados anteriormente?
A. commands
B. review
C. history
D. export

39.Que tecla permite ver el último comando ejecutado?
A. down arrow
B. up arrow
C. right arrow
D. left arrow

40.Que comando te permite usar el editor por defecto para editar varios comandos del archivo historico?
A. history
B. edit
C. fc
D. view

41.Es posible ejecutar comandos que no están en el PATH, si conoces la ruta completa y el nombre del comando
A. Verdadero
B. Falso

42.¿Cual de las siguientes respuestas enviará los datos de la salida del comando ls al archivo myfiles?
A. ls | myfiles
B. ls > myfiles
C. ls < myfiles
D. ls | xargs myfiles

43.¿Como se redirecciona stdout y stderr a un archivo?
A. <&
B. >&
C. |&
D. &&

44.¿Que respuesta ordenaría alfabéticamente el fichero mylist, numeraría las lineas y finalmente lo separaría en ficheros que contuvieran 60 lineas cada uno?
A. sort mylist | nl > -60 lists
B. sort mylist > nl > split -60 > lists
C. sort mylist | nl | split -60 lists
D. sort mylist | nl | tee lists | split -60 lists

45.La utilidad _____ facilita información para resolver problemas, guardando la salida de un comando que es usado en tubería (pipe) con otro?

46.¿Cual será la salida del comando ps -ae?
A. Sólo se muestran los procesos para el usuario actual, menos la sesión principal.
B. Sólo se muestran los procesos para el usuario actual, incluida la sesión principal.
C. Se muestran todos los procesos de todos los usuarios.
D. Se muestran todos los procesos de todos los usuarios, menos cualquier sesión principal.

47.¿Cual de los siguientes comandos, produce el mismo resultado que el comando ps -e?
A. ps -f
B. ps -A
C. ps -l
D. ps -u

48.¿Cual de los siguientes comandos ejecuta un proceso en segundo plano?
A. -
B. +
C. %
D. &

49.Estando a punto de dejar la oficina por hoy, hace un minuto que se lanzó un trabajo de compilación que puede durar horas en segundo plano. Cuando ésta compilación acabe, se necesita que otro proceso se ejecute para imprimir unos resultados.
¿Cual de los siguientes comandos se usará para realizar estas tareas?
A. bg {proceso}
B. fg {proceso}
C. wait $! ; {proceso}
D. sleep ; {proceso}

50.¿Cual de los siguientes, representa al trabajo más reciente, lanzado en segundo plano, tal como muestra el comando jobs?
A. [1] Running {proceso}
B. [1]- Running {proceso}
C. [1]% Running {proceso}
D. [1]+ Running {proceso}

51.Se quiere ver un listado completo de todos los procesos que hay actualmente en ejecucion, y guardar una copia en un fichero llamado 'procesos'. ¿Que comando, o conjunto de comandos se usaría para esto?
A. ps -f ; ps -f > procesos
B. ps -ef >> procesos
C. ps -ef | tee procesos
D. ps -ef ; tee procesos

52.¿Que comando se utiliza para cambiar la prioridad de un proceso en ejecución?.

53.¿Que comando se utiliza para establecer la prioridad de un proceso en el momento de su arranque?.
A. jobs
B. renice
C. nice
D. top

54.Se está trabajando con vi en un fichero que necesita modificaciones importantes. El cursor está en la primera letra de una frase que dura tres líneas. Que comando se puede usar para borrar toda la frase en cuestión?
A. dd
B. d$
C. d)
D. dw

55.Mary ha abierto un fichero log desde vi y necesita moverse a la última línea para ver la entrada más reciente. Que combinación de teclas debe introducir para conseguirlo?
A. G
B. X
C. ZZ
D. $

56.Cual de las siguientes combinaciones de teclas se pueden usar para salir del editor vi?
A. :w
B. :q
C. :q!
D. ZZ

57.Evan tiene el cursor en la mitad de una línea y necesita cambiar el fichero desde donde está el cursor hasta el final de la línea. Que secuencia debería usar para decirle al editor vi lo que quiere hacer?
A. c^
B. C
C. .c}
D. .r

58.Qué secuencia se usa para saber en que número de línea se encuentra situado el cursor?
A. cw
B. /
C. .Ctrl+F
D. . Ctrl+G

59.Estás editando un fichero y quieres buscar el número de empleado "12345". Este número puede aparecer por muchos sitios del fichero, pero se refiere al número de empleado sólo si son los primeros 5 caracteres de la línea. Cómo indicamos la búsqueda correcta?

60.Estás editando un fichero y quieres buscar "home". Este texto puede aparecer en muchos sitios, pero significa que es el empleado sólo si estos cuatro caracteres son los primeros dentro de la palabra. Que se debe introducir para iniciar una búsqueda correcta?

61.Qué secuencia se debe usar para copiar ocho líneas en el buffer para poder ser copiadas en otra localización dentro del fichero?
A. 8cc
B. cc8
C. 8pp
D. .pp8
E. 8yy
F. yy8

62.Estás al final de un fichero muy largo y quieres buscar hacia atrás la ciudad "Muncie". Que secuencia usarías para buscar hacia atrás y encontrar la primera coincidencia con esta palabra?
A. /Muncie
B. :Muncie
C. .Ctrl+F
D. ?Muncie

63.En muchas versiones de Linux, que editor se incluye que arranca automáticamente -y transparentemente - al teclear vi?
A. vim
B. emacs
C. de
D. edlin

Escenarios

1. Necesitas imprimir el archivo grande que se ha creado en el ejercicio anterior de forma que cada pagina esté numerada y tenga una cabecera que contenga el nombre de tu empresa y la fecha actual. ¿Como realizarías esta tarea?

2. Tu empresa almacena un gran número de pequeños archivos en un directorio dentro de un sistema Linux. Se necesita combinar los contenidos de estos archivos en un gran archivo. Posteriormente se necesita partir ese gran archivo en varios archivos de la misma longitud. ¿Como se podría hacer?

RESPUESTAS PRE-TEST

1. El comando locate utiliza la base de datos slocate para encontrar los ficheros.
2. El kernel del sistema se encuentra en el directorio /boot.
3. Los soft links pueden apuntar a diferentes sistemas de ficheros.
4. El comando ls se utiliza para listar los ficheros de un directorio.
5. El comando mkdir se utiliza para crear un directorio.
6. El comando touch puede utilizarse para crear un fichero vacío.
7. El comando whereis se utiliza para buscar un comando en la declaración PATH.
8. El comando find busca recursivamente un fichero en un directorio.
9. La utilidad nl se usa para numerar las líneas de un archivo.
10. La utilidad join se usa para juntar líneas de diferentes archivos.
11. El comando fmt se usa para dar formato con lineas de igual longitud a un archivo.
12. El comando tac muestra por pantalla un archivo de forma inversa, lo contrario que cat.
13. Las utilidades tr y sed son usadas para borrar y substituir caracteres en un archivo.
14. La utilidad od se usa para ver archivos en forma octal, hexadecimal, y otros formatos.
15. La utilidad sort se usa para ordenar el contenido de un archivo en orden numérico o alfabético.
16. El fichero /etc/passwd contiene el shell por defecto para cada usuario.
17. Añadiendo a la variable PATH editando el fichero /etc/profile, añadira el PATH a todos los usuarios del sistema.
18. La variable HISTSIZE indica el numero de entradas a almacenar en el fichero .bash_history
19. La tecla TAB presionada una vez, y la tecla ESC presionada dos veces completa los comandos, si solo hay un posible comando lo iguala.
20. El comando fc permite editar las entradas desde el fichero ~/.bash_history usando el editor por defecto del sistema.
21. El fichero .bash_history, localizado en el directorio home de cada usuario, contiene el historial de comandos.
22. Se pueden ver las variables usando el comando echo, como echo $HOME.
23. Para ejecutar un comando desde pwd, escribe el comando precedido de ./, como en ./commandname
24. La sustitución de comandos permite la salida de un comando en lugar del nombre del comando.
25. La opción -R permite funciones recursivas de un comando.
26. El carácter “mayor que” (>) se usa para redireccionar la salida standard (stdout) hacia un archivo.
27. Los caracteres >& se usan para redireccionar la salida standard (stdout) y de errores (stderr) hacia un archivo.
28. La utilidad tee se usa para enviar los resultados de la ejecución de un comando a la salida standard (stdout) y a un archivo al mismo tiempo.

RESPUESTAS TEST

1. **D.** El path absoluto al comando ls es /bin/ls. Para más información mira la sección "Manejando ficheros"
2. **ls a*.** El comodín se utiliza para especificar cualquier carácter o conjunto de caracteres. Este ejemplo listaría los ficheros y directorios llamados 'a' o cuyo nombre comience por 'a'. Para más información mira la sección "Listando el contenido del directorio".
3. **B.** Pueden crearse ficheros con el comando touch si el nombre de fichero utilizado no existiese. Si existiese, simplemente se cambiaría la hora de acceso.
   Para más información mira la sección "Cambiando marca de hora en ficheros".
4. **C.** El nombre y la ubicación de los ficheros se cambia con el comando mv. Para más información mira la sección "Moviendo ficheros".
5. **dd.** El comando dd, o volcado de directorio, se utiliza para convertir y copiar ficheros. Para más información mira la sección "dd".
6. **C.** El kernel del sistema se encuentra en el directorio /boot. Para más información mira la sección "Directorios del sistema".
7. **A.** Los ficheros de configuración del sistema se encuentran en el directorio /etc. Para más información mira la sección "Ubicación de los ficheros estándar".
8. **B.** El directorio home del usuario root es /root. Para más información mira la sección "Directorios del sistema".
9. **B.** Los ficheros de correo se encuentran en el directorio /var/spool. Para más información mira la sección "Directorios del sistema".
10. **D.** La base de datos slocate se actualiza con el comando updatedb. Para más información mira la sección "locate".
11. **which.** El comando which muestra el path del comando especificado. Para más información mira la sección "which".
12. D. Cuando se copia un soft link, se crea una nueva copia del fichero original en el destino especificado. Para más información mira la sección "Enlaces simbólicos".
13. D. Cuando se usa direccionamiento con la utilidad sed, los números de línea y los rangos buscados se especifican antes del comando "s". El rango va separado por una coma.
14. A y D. La opción -l se usa para especificar el numero de líneas que contendrá cada archivo tras el uso de split. En cualquier caso, cuando no se le da ninguna opción, el numero indicado se asume como el numero de lineas.
15. B y C. Las utilidades paste y join se usan para combinar lineas de un archivo. El comando split se usa para dividir un archivo en trozos. La utilidad cut borra texto de un archivo.
16. D. El comando tail se usa para ver el final de un archivo y la opción -5 indica el numero de lineas que se quiere ver.
17. A y B. La utilidad od muestra los ficheros en octal por defecto. La opción -t seguida de o, se usa para indicar el formato octal.
18. D. Las tuberías se usan para enviar datos de un comando a otro, por lo tanto A y B son incorrectas. La respuesta C no crea dos archivos, así pues solo la D es la respuesta correcta.
19. tac. La utilidad tac se usa para ver un fichero comenzando desde la ultima linea hasta la primera.
20. C. La utilidad wc se usa para conocer los totales de un archivo, bien sean líneas, palabras o bytes.
21. expand. La utilidad expand se usa para convertir las tabulaciones (el carácter "tab") en espacios.
22. C. Con fmt se intenta crear líneas de igual tamaño en un fichero.
23. D. Usamos tr para borrar o substituir caracteres en un archivo.
24. A. Hay que usar la opción -c con sort para verificar que el contenido del fichero se ha ordenado.

25.C. El shell por defecto usado en los sistemas Linux es bash. Ver la sección “Entendiendo Shells” para mas información.
26.D. El fichero /etc/shells contiene una lista de los shells disponibles. Ver la sección de “Entendiendo Shells”.
27.B. El comando chsh se usa para ver y cambiar la configuración del shell. Ver la sección “Entendiendo Shells” para mas información.
28.D. Las opciones del comando preceden a los argumentos cuando se entran en la linea de comandos. Ver la sección “Usando la linea de comandos” para mas información.
29.B. El punto y coma (;) se usa para separar multiples comandos en una sola linea. Ver la sección “Usando la linea de comandos” para mas información.
30.B. La tecla \ causa que cualquier tecla que le siga sea ignorada. Esto permite a la tecla Enter ser ignorada y que continue el comando de la linea siguiente. Ver la sección “Usando la linea de comandos” para mas información.
31.A. La tecla TAB completa comandos cuando se pulsa una vez. Ver la sección “completar comandos” para mas información.
32.B. La tecla escape completa comandos cuando se pulsa dos veces. Ver la sección “completar comandos” para mas información.
33.D. El fichero /etc/profile almacena la declaración de las variables de todo el sistema usando el shell bash. Ver la sección “Entendiendo Shells” para mas información.
34. ./update_mozilla. Cuando ejecutamos un comando desde pwd, el nombre del comando se precede por un punto y una barra (./), que especifica que el comando esta en el directorio actual. Ver la sección “Editando la variable path” para mas información.
35.C. El comando export causa cambios en una variable de entorno para que sea accesible por el shell del usuario. Ver la sección “Variables de entorno y configuraciones” para mas información.
36.A. La variable PS1 se usa para hacer cambios al prompt. Ver la sección “Prompt” para mas información.
37.C. El fichero /etc/passwd contiene la asignación de directorios del home del usuario. Ver la sección “variables de entorno y configuraciones” para mas información.
38.C. El comando history muestra los comandos introducidos previamente que se encuentran almacenados en el fichero del usuario .bash_history. Ver la sección “”Usando el fichero history” para mas información.
39.B. La flecha hacia arriba se usa para pasearse por los comandos introducidos previamente. Ver la sección “Usando el fichero history” para mas información.
40.C. El comando fc permite editar los comandos introducidos previamente usando el editor por defecto. Ver la sección “fc” para mas información.
41.A. Los comandos se pueden ejecutar usando el path completo si el usuario tiene los permisos necesarios. Ver la sección “Editando la variable PATH” para mas información.
42.A. Las tuberías se usan para enviar datos de la salida de un comando a la entrada de otro.
43.B. Los caracteres >& envían la salida standard y de errores (stdout y stderr) a un archivo.
44.D. Las tuberías se usan para enviar datos de un comando a otro, por lo tanto A y B son incorrectas. La respuesta C no crea dos archivos, así pues solo la D es la respuesta correcta.
45. tee. Con tee se envía la salida de un comando a un archivo y a la salida estandar (stdout) al mismo tiempo.
46.La respuesta correcta es la c. La opción -a, cuando se usa a solas, no muestra la sesión principal, pero con la opción -e se muestra todo. Es equivalente a la opción 'mostrar selectivamente' y 'mostrar todo'. Se pueden poner las dos opciones, pero la opción -e gana.
47. La respuesta correcta es la b. La opción -e se usa para mostrar cualquier cosa, y la opción -A se usa para mostrar todo. La opción -f (respuesta a) mostrará un listado completo, pero sólo para el usuario actual. La opción -l (respuesta c) mostrara un listado completo, pero de nuevo, sólo para el usuario actual. La opción -u (respuesta d), mostrara la información relativa al usuario.

48. La respuesta correcta es la d. El ampersand (&) lanza un trabajo en segundo plano. Las otras opciones se ignoran al ser usadas como carácter final en la linea de comandos.
49. La respuesta correcta es la c. El signo de dolar y el de admiración ($!) representan al último proceso ejecutado en segundo plano. Cuando se usan con el comando wait (espera), el proceso en segundo plano deberá acabar antes que el próximo proceso comience a ejecutarse. Todas las otras opciones son incorrectas para especificar que un proceso debe esperar a otro para ejecutarse.
50. La respuesta correcta es la d. El signo más (+) a la derecha del numero de trabajo indica el trabajo más reciente puesto en segundo plano. Las otras opciones son invalidas dado que no tienen nada que ver con el trabajo más reciente puesto en segundo plano.
51. La respuesta correcta es la c. El comando tee coge los datos que recibe y lo muestra por pantalla, y a la vez, lo guarda en un fichero. La respuesta a no muestra todos los procesos, y requiere dos procesos, en contra de uno sólo de la opción c. La respuesta b guarda los procesos en el fichero pero no lo muestra por pantalla. La opción d tiene una sintaxis incorrecta.
52. El comando renice se utiliza para cambiar la prioridad de un proceso que está ejecutándose en el sistema.
53. C. El comando nice se utiliza para cambiar el nivel de prioridad de un proceso en el momento de su arranque.
54. La respuesta correcta es c, que borrará toda la frase. La respuesta a borra la línea, mientras que la respuesta b borra desde el cursor hasta el final de la línea. La d solo borra la primera palabra.
55. La respuesta correcta es a porque el comando G sitúa el cursor al final del fichero. La respuesta b es incorrecta porque X borra el carácter antes del cursor. La respuesta c es incorrecta porque ZZ guarda el fichero y sale del editor. La respuesta d es incorrecta porque $ significa final de línea , pero no de fichero.
56. Las respuestas correctas a esta pregunta son b, c y d. La respuesta b vale, pero solo se puede usar si no se han hecho cambios al fichero. La respuesta c vale y puede usarse si se han hecho cambios pero no se desea guardarlos. La respuesta d guarda el fichero y sale del editor. La respuesta a es incorrecta porque guarda el fichero, pero no sale del editor.
57. La respuesta correcta es b, que indica que los cambios se han de hacer desde la situación actual del cursor hasta el final de la línea. La respuesta a es incorrecta porque indica que los cambios se hagan desde la localización actual hasta el principio de la línea. La respuesta c es incorrecta porque indica que se hagan los cambios en el resto del párrafo. La respuesta d es incorrecta porque permite cambiar sólo un carácter.
58. La respuesta correcta es d, que muestra la línea actual y otra información al final de la pantalla. La respuesta a es incorrecta porque es para cambiar una palabra. La respuesta b es incorrecta porque es para cambiar al modo de búsqueda. La respuesta c es incorrecta porque permite moverse por el fichero hasta la siguiente pantalla.
59. La respuesta correcta es “/^12345“
    El “/“ se necesita para introducir cadenas de búsqueda, y el carácter “ ^ “ significa que la cadena debe aparecer al principio de la línea para ser relevante.
60. La respuesta correcta es “ / \ < home“
    El “ / “ se necesita para introducir cadenas de búsqueda, y la secuencia “ \ < “ identifica que el texto debe estar al principio de la línea.
61. La respuesta correcta es e, que copia las siguientes ocho líneas en el buffer. La respuesta a es incorrecta porque permite cambiar (no copiar) las siguientes ocho líneas. La respuesta b es incorrecta porque usa una sintaxis incorrecta y no funciona. La respuesta c tampoco funcionaria porque no existe dicho comando – mismo problema en la respuesta d. La respuesta f falla porque solo extrae una línea. Haría “algo“ 8 veces – dependiendo de lo que se escriba después del ocho.

62.La respuesta correcta es d, la cual busca hacia atrás por el fichero hasta que encuentra la primera coincidencia. La respuesta a busca hacia adelante desde la situación del cursor para encontrar la primera coincidencia. La respuesta b permite entrar un comando (los dos puntos), y recibe el comando desconocido de "Muncie" y devuelve un error. La respuesta c simplemente mueve el cursor a la siguiente pantalla y no realiza ninguna búsqueda.

63.La respuesta correcta es a, que es una versión mejorada de vi. La respuesta b es un editor que se puede bajar gratuitamente y usar en lugar de vi. La respuesta c es un viejo editor que existía en Unix y que era casi imposible trabajar con el. La respuesta d es un viejo editor de los primeros días del DOS y solo era ligeramente más fácil de usar.

Respuestas Escenarios

1. pr -h "Nombredelaempresa" largefile.
   El comando pr automáticamente imprime la fecha y el numero de pagina como parte de la información de cabecera. Para añadir el nombre de la empresa la opción -h es la correcta, dándole lo que se desea incluir entre dobles comillas. El fichero, largefile, es el nombre del que creamos en el escenario anterior, que ahora imprimimos con el comando pr.
2. ls | xargs cat | tee largefile | split -60
   La utilidad xargs se usa para trabajar con grandes cantidades de datos. El comando tee envía los datos a un fichero y a la salida standard. Finalmente los datos son divididos en paginas de 60 lineas cada una.

Bibliografía y enlaces recomendados

LPIC 1 Certification Bible (Bible) by Angie Nash, Jason Nash
John Wiley & Sons; Bk&CD-Rom edition (July 1, 2001) ISBN: 0764547720

LPI Linux Certification in a Nutshell by Jeffrey Dean
O'Reilly & Associates; 1st ed edition (May 15, 2001) ISBN: 1565927486

CramSession's LPI General Linux Part 1 : Certification Study Guide
CramSession.com; ISBN: B000079Y0V; (August 17, 2000)

Referencias Unix Reviews
http://www.unixreview.com/documents/s=7459/uni1038932969999/

Página LPI: www.lpi.org

Apuntes IBM: http://www-106.ibm.com/developerworks/edu/l-dw-linux-lpir21-i.html

Manuales GPL: http://www.nongnu.org/lpi-manuals/