**1**Bash and Bash Scripts

Hoy en día, muchas personas usan computadoras con una interfaz gráfica de usuario (GUI) como GNOME. Aunque las GUI ofrecen muchas características, están limitadas cuando realizan la ejecución de tareas automatizadas. Los shells complementan bien las GUI, y este capítulo ofrece una descripción general de algunos aspectos de los shells, en este caso, el shell Bash

**1.1** Qué es “la Shell”?

Tradicionalmente, el shell de Linux es Bash (Bourne nuevamente Shell). Cuando este capítulo habla de “la concha” se refiere a Bash. Hay más shells disponibles (ash, csh, ksh, zsh, …), cada uno de los cuales emplea diferentes funciones y características. Si necesita más información sobre otros shells, busque shell en aptitude

**1.1.1**Bash Configuration Files

Una “Shell” puede ser invocada como:

1. **Shell de inicio de sesión interactivo.** Esto se usa al iniciar sesión en una máquina, al invocar Bash con la opción --login o al iniciar sesión en una máquina remota con SSH.
2. **Shell interactivo "ordinario"**. Este suele ser el caso al iniciar xterm, konsole, gnome-terminal o herramientas de interfaz de línea de comandos (CLI) similares.
3. **Shell no interactivo**. Esto se invoca cuando se invoca un script de shell en la línea de comandos.

Según el tipo de shell que utilice, se leerán diferentes archivos de configuración. Las siguientes tablas muestran los archivos de configuración de shell de inicio de sesión y de no inicio de sesión

**TABLE 1.1:**ARCHIVOS DE CONFIGURACIÓN DE BASH PARA shell DE INICIO DE SESIÓN

| **Archivo** | **Descripción** |
| --- | --- |
| /etc/profile | No modificar este fichero. Si se hace, los cambios no serán guardados en el siguiente inicio de sesión. |
| /etc/profile.local | Usado para extender el perfil almacenado en /etc/profile |
| /etc/profile.d/ | Contiene archivos de configuración de todo el sistema para programas específicos |
| ~/.profile | Inserte la configuración específica del usuario para los shells de inicio de sesión aquí |

Tenga en cuenta que el shell de inicio de sesión también genera los archivos de configuración enumerados en la Tabla 1.2, “Archivos de configuración de Bash para shells que no son de inicio de sesión”

.

**TABLE 1.2:**Archivos de configuración para shells sin login (automatizadas)

|  |  |
| --- | --- |
| /etc/bash.bashrc | No modificar este fichero. Si se hace, los cambios no serán guardados en el siguiente inicio de sesión.! |
| /etc/bash.bashrc.local | Use este archivo para insertar sus modificaciones Bash en todo el sistema local. |
| ~/.bashrc | Para insertar la configuración específica del usuario. |

Adicionalmente, Bash usa más ficheros de configuración:

**TABLE 1.3:**Archivos de configuración especiales para bash:

| **File** | **Description** |
| --- | --- |
| ~/.bash\_history | Contiene la lista de comandos escritos. El número de comandos depende de la configuración (normalmente 1000 o 2000 últimos comandos) |
| ~/.bash\_logout | Se ejecuta en los cierres de sesión. |
| ~/.alias | Alias definidos por el usuario para los comandos usados frecuentemente. Para más información se puede ejecutar el comando:  $ see man 1 alias |

Bash\_history

Creando alias

Aunque más adelante se hace referencia a la creación de alias, vamos a ver su posible utilidad:

* 1. Podemos crear un alias llamado 1l para el comando ls –la
     1. alias lL="ls -la"
  2. Si queremos quitar el alias, escribimos::
     1. unalias 1l
  3. Si queremos un alias persistente, lo escribiremos en bash.rc al final del archivo, por ejemplo pondremos:
     1. #########
     2. # Aliases
     3. #########
     4. alias ll="ls -lhA"

**Ejercicio**: a continuación pongo uns cuantos ejemplos de alias que podrían ser útiles. Busca qué hacen cada uno de ellos y si te parecen útiles agrégalos a **~/.bashrc**

1. $ alias ls="ls -CF"
2. $ alias sl="ls"
3. alias lsl="ls -lhFA | less"
4. alias cd..="cd .."
5. alias ..="cd .."
6. alias fhere="find . -name "

No-Login Shells (shells sin login):

Hay shells especiales que impiden que los usuarios inicien sesión en el sistema: /bin/false y /sbin/nologin. Ambos fallan silenciosamente cuando el usuario intenta iniciar sesión en el sistema. Esto fue pensado inicialmente como una medida de seguridad, aunque los sistemas operativos Linux modernos tienen herramientas más efectivas para controlar el acceso al sistema, como PAM y AppArmor.

El valor predeterminado en Ubuntu y otros sistemas modernos es asignar /bin/bash a los usuarios humanos y /bin/false o /sbin/nologin a los usuarios del sistema.

El usuario **none** tiene /bin/bash por razones históricas, ya que es un usuario con privilegios mínimos que solía ser el predeterminado para los usuarios del sistema. Sin embargo, la pequeña cantidad de seguridad que se gana al usar **none** se pierde cuando varios usuarios del sistema la usan. Debería ser posible cambiarlo a /sbin/nologin; la forma más rápida de probarlo es cambiarlo y ver si rompe algún servicio o aplicación.

Use el siguiente comando para enumerar qué shells están asignados a todos los usuarios, usuarios humanos y del sistema, en /etc/passwd. La salida varía según los servicios y usuarios de su sistema:

$ sort -t: -k 7 /etc/passwd | awk -F: '{print $1"\t" $7}' | column -t

tux /bin/bash

nobody /bin/bash

root /bin/bash

avahi /bin/false

chrony /bin/false

dhcpd /bin/false

dnsmasq /bin/false

ftpsecure /bin/false

lightdm /bin/false

mysql /bin/false

postfix /bin/false

rtkit /bin/false

sshd /bin/false

tftp /bin/false

unbound /bin/false

bin /sbin/nologin

daemon /sbin/nologin

ftp /sbin/nologin

lp /sbin/nologin

mail /sbin/nologin

man /sbin/nologin

nscd /sbin/nologin

polkitd /sbin/nologin

pulse /sbin/nologin

qemu /sbin/nologin

radvd /sbin/nologin

rpc /sbin/nologin

statd /sbin/nologin

svn /sbin/nologin

systemd-coredump /sbin/nologin

systemd-network /sbin/nologin

systemd-timesync /sbin/nologin

usbmux /sbin/nologin

vnc /sbin/nologin

wwwrun /sbin/nologin

messagebus /usr/bin/false

scard /usr/sbin/nologin

**1.1.2**La estructura de directorios:

La siguiente tabla proporciona una breve descripción general de los directorios de nivel superior más importantes que encontrará en un sistema Linux..

**TABLE 1.4:**Vista general de un arbol raiz de directorio en linux

| **Directory** | **Contents** |
| --- | --- |
| / | Root directory—El punto de comienzo del árbol de directorio. |
| /bin | Archivos binarios esenciales, como comandos que necesitan tanto el administrador del sistema como los usuarios normales. Por lo general, también contiene las Shell, como Bash |
| /boot | Archivos estáticos del gestor de arranque |
| /dev | Archivos necesarios para acceder a dispositivos específicos del host. |
| /etc | Archivos de configuración del sistema específicos del host. |
| /home | Contiene los directorios de inicio de todos los usuarios que tienen cuentas en el sistema. Sin embargo, el directorio de inicio de root no se encuentra en /home sino en /root. |
| /lib | Bibliotecas compartidas esenciales y módulos del núcleo. |
| /media | Puntos de montaje para medios extraíbles. |
| /mnt | Punto de montaje para montar temporalmente un sistema de archivos. |
| /opt | Paquetes de software de aplicaciones complementarias. |
| /root | Directorio de inicio para el superusuario root. |
| /sbin | Binarios esenciales del sistema. No accesibles por usuarios sin privilegios administrativos. |
| /srv | Datos de los servicios prestados por el sistema. |
| /tmp | Ficheros temporales. |
| /usr | Jerarquía secundaria con datos de solo lectura. |
| /var | Datos variables como archivos de registro. |
| /windows | Solo disponible si tiene Microsoft Windows\* y Linux instalados en su sistema. Contiene los datos de Windows. |

La siguiente lista proporciona información más detallada y da algunos ejemplos de qué archivos y subdirectorios se pueden encontrar en los directorios:

/bin

Contiene los comandos básicos de shell que pueden ser utilizados tanto por root como por otros usuarios. Estos comandos incluyen ls, mkdir, cp, mv, rm y rmdir. /bin también contiene Bash, el shell predeterminado.

/boot

Contiene datos necesarios para el arranque, como el cargador de arranque, el kernel y otros datos que se utilizan antes de que el kernel comience a ejecutar programas en modo de usuario.

/dev

Contiene archivos de dispositivos que representan componentes de hardware.

/etc

Contiene archivos de configuración locales que controlan el funcionamiento de programas como el sistema X Window. El subdirectorio /etc/init.d contiene secuencias de comandos de inicio LSB que se pueden ejecutar durante el proceso de arranque.

/home/*USERNAME*

Contiene los datos privados de cada usuario que tiene una cuenta en el sistema. Los archivos que se encuentran aquí solo pueden ser modificados por su propietario o por el administrador del sistema. De forma predeterminada, su directorio de correo electrónico y la configuración de su escritorio personal se encuentran aquí en forma de archivos y directorios ocultos, como .gconf/ y .config.

**Note: Directorio home en un entorno de red**

Si está trabajando en un entorno de red, su directorio de inicio puede asignarse a un directorio en el sistema de archivos que no sea /home.

/lib

Contiene las bibliotecas compartidas esenciales necesarias para iniciar el sistema y ejecutar los comandos en el sistema de archivos raíz.

/media

Contiene puntos de montaje para medios extraíbles, como CD-ROM, discos flash y cámaras digitales (si utilizan USB). /media generalmente contiene cualquier tipo de unidad excepto el disco duro de su sistema. Cuando su medio extraíble se haya insertado o conectado al sistema y se haya montado, puede acceder a él desde aquí

/mnt

Este directorio proporciona un punto de montaje para un sistema de archivos montado temporalmente. root puede montar sistemas de archivos aquí.

/opt

Reservado para la instalación de software de terceros. El software opcional y los paquetes de programas complementarios más grandes se pueden encontrar aquí.

/root

Directorio de inicio para el usuario root. Los datos personales de root se encuentran aquí.

/run

Un directorio tmpfs utilizado por systemd y varios componentes. /var/run es un enlace simbólico a /run.

/sbin

Como indica la s, este directorio contiene utilidades para el superusuario. /sbin contiene los binarios esenciales para arrancar, restaurar y recuperar el sistema además de los binarios en /bin.

/srv

Contiene datos para los servicios proporcionados por el sistema, como FTP y HTTP.

/tmp

Este directorio es utilizado por programas que requieren almacenamiento temporal de archivos..

**Importante: Limpieza de /tmp en el momento del arranque**

No se garantiza que los datos almacenados en /tmp sobrevivan a un reinicio del sistema. Depende, por ejemplo, de la configuración realizada en /etc/tmpfiles.d/tmp.conf..

/usr

/usr no tiene nada que ver con los usuarios, pero es el acrónimo de recursos del sistema Unix. Los datos en /usr son datos estáticos de solo lectura que se pueden compartir entre varios hosts que cumplen con el estándar de jerarquía del sistema de archivos (FHS). Este directorio contiene todos los programas de aplicación, incluidos los escritorios gráficos como GNOME, y establece una jerarquía secundaria en el sistema de archivos. /usr contiene varios subdirectorios, como /usr/bin, /usr/sbin, /usr/local y /usr/share/doc..

/usr/bin

Contiene programas de acceso general..

/usr/sbin

Contiene programas reservados para el administrador del sistema, como funciones de reparación

/usr/local

En este directorio, el administrador del sistema puede instalar extensiones locales independientes de la distribución.

/usr/share/doc

Contiene varios archivos de documentación y las notas de la versión de su sistema. En el subdirectorio manual encontrará una versión en línea de este manual. Si se instala más de un idioma, este directorio puede contener versiones de los manuales para diferentes idiomas.

En paquetes, busque la documentación incluida en los paquetes de software instalados en su sistema. Para cada paquete, se crea un subdirectorio /usr/share/doc/packages/PACKAGENAME que a menudo contiene archivos README para el paquete y, a veces, ejemplos, archivos de configuración o scripts adicionales.

Si los HOWTO están instalados en su sistema, /usr/share/doc también contiene el subdirectorio howto en el que puede encontrar documentación adicional sobre muchas tareas relacionadas con la configuración y el funcionamiento del software de Linux.

/var

Mientras que /usr contiene datos estáticos de solo lectura, /var es para datos que se escriben durante el funcionamiento del sistema y, por lo tanto, son datos variables, como archivos de registro o datos en cola. Para obtener una descripción general de los archivos de registro más importantes que puede encontrar en /var/log.

**1.1.3 Caracteres especiales:**

Los caracteres <, >, |, & \*, ? , ~, [, ], {, } son ejemplos de caracteres especiales para Bash.

Más adelante veremos otros comandos tienen sus propios caracteres especiales, como puedan ser los comandos que usan expresiones regulares o los operadores de manejo de cadenas.

La Tabla siguiente muestra todos los caracteres especiales de Bash

|  |  |
| --- | --- |
| Carácter | Descripción |
| ~ | Directorio home |
| ` | Sustitución de comando (obsoleto) |
| $() | Sustitución de comando (actual) |
| # | Comentario |
| $ | Variable |
| & | Proceso en background |
| ; | Separador de comandos |
| \* | Comodín 0 a n caracteres |
| ? | Comodín de un sólo carácter |
| / | Separador de directorios |
| ( | Empezar un subshell |
| ) | Terminar un subshell |
| \ | Carácter de escape |
| < | Redirigir la entrada |
| > | Redirigir la salida |
| | | Pipe |
| [ | Empieza conjunto de caracteres comodín |
| ] | Acaba conjunto de caracteres comodín |
| { | Empieza un bloque de comando |
| } | Acaba un bloque de comando |
| ' | Entrecomillado fuerte |
| " | Entrecomillado débil |
| ! | No lógico de código de terminación |

Esta tabla conviene tenerla muy presente cuan comenzamos a practicar con scripts y a interpretarlos.

**1.1.4 Entrecomillado**

Aunque los caracteres especiales son muy útiles para Bash, a veces queremos usar un carácter especial literalmente, es decir sin su significado especial, en este caso necesitamos entrecomillarlo (quoting).

Por ejemplo si queremos escribir en consola el mensaje: 2\*3>5 es una expresión cierta, podemos usar el comando echo así:

*$ echo 2\*3>5 es una expresion cierta*

La interpretación de lo escrito quizá difiera de lo que queríamos hacer. Se ha creado el fichero 5 con el texto 2\*3 es una expresión cierta. La razón está en que >5 ha sido entendido como redirigir al fichero 5, y además se ha intentado ejecutar el carácter especial \*, pero al no encontrar ningún fichero que cumpliera el patrón no se ha expandido y se ha pasado el parámetro a echo tal cual.

Sin embargo si entrecomillamos usando ', el carácter de entrecomillado fuerte obtenemos el resultado esperado:

*$ echo '2\*3>5 es una expresion cierta'*

2\*3>5 es una expresion cierta

Un ejemplo más práctico del entrecomillado es el comando find que nos permite buscar ficheros por muchos criterios. Por ejemplo para buscar por nombre usa el argumento -name patron. Si por ejemplo queremos buscar todos los ficheros .c en nuestra máquina podemos intentar hacer:

*$ find / -name \*.c*

Pero si tenemos la mala suerte de que en el directorio actual exista algún fichero .c, sustituirá el comodín y buscará el nombre de el/los ficheros de nuestro directorio actual en el resto disco duro. Para evitarlo es recomendable entrecomillarlo:

*$ find / -name '\*.c'*

**Hagamos la prueba:**

Nos aseguramos que en la carpeta actual no hay ningún archivo .c

Ejecutamos en nuestro Linux

*$ find / -name \*.c*

En los primeros resultados no debería aparecer la ruta en la que nos encontramos y ningún archivo .c en dicha ruta. Si es así, el comando find ha funcionado correctamente. Al no hacerlo con sudo, habrá muchas líneas de error que contienen “permiso denegado”. Vamos a quitarlas modificando la salida de error a null para que desaparezcan. Ejecutamos:

*$ find 2>null / -name \*.c*

La salida ya queda limpia y los errores no aparecen al haber sido redirigidos.

Ahora nos vamos a nuestra carpeta scripts (recordad /home/alumno/scripts) y en ella creamos con touch un archivo .c ejecutamos:

*$ cd $HOME/scripts*

*$ touch kk.c*

Y acto seguido nuestra búsqueda:

*$ find 2>null / -name \*.c*

Observa como cambia la búsqueda y se reduce a un resultado. El archivo .c contenido en tu carpeta.

En la Tabla 1.2 aparece el entrecomillado débil ", el cual pasa sólo por algunos de los pasos del shell, o dicho con otras palabras, interpreta sólo algunos caracteres especiales. Más adelante veremos cuando es preferible usar este tipo de entrecomillado, de momento usaremos sólo el entrecomillado fuerte.

**1.1.5 Caracteres de escape**

Otra forma de cambiar el significado de un carácter de escape es precederlo por \, que es lo que se llama el carácter de escape.

Por ejemplo en el ejemplo anterior podríamos haber hecho:

$ echo 2\\*3\>5 es una expresion cierta

2\*3>5 es una expresion cierta

Donde hemos puesto el carácter de escape a los caracteres especiales para que Bash no los interprete.

También este carácter se usa para poder poner espacios a los nombres de ficheros, ya que el espacio es interpretado por Bash como un separador de argumentos de la línea de comandos, si queremos que no considere el espacio como un cambio de argumento podemos preceder el espacio por \.

Por ejemplo el fichero Una historia.avi lo reproduzco en mi ordenador con mplayer así:

$ mplayer Una\ historia.avi

Si queremos que Bash no interprete el carácter de escape podemos entrecomillarlo '\', o bien hacer que él se preceda a si mismo \\.

**1.1.6 Entrecomillar los entrecomillados**

Podemos usar el carácter de escape para que no se interpreten los entrecomillados simples o dobles, es decir:

$ echo \"2\\*3\>5\" es una expresion cierta

"2\*3>5" es una expresion cierta

Otra manera

$ echo \'2\\*3\>5\' es una expresion cierta

'2\*3>5' es una expresion cierta

Funcionmiento de las comillas:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| “ | Comillas dobles | Las comillas dobles ("double quote") protegen todo lo encerrado entre dos comillas dobles, excepto $, ', " y \. Use las comillas dobles cuando solo desee variables y sustitución de comandos..   |  | | --- | | \* **Variable** - Yes | | \* Comodine**s** - No | | \* **Sustitución de comandos** - yes | |  |
| ‘ | Comillas simpples | La comilla simple ('simple quote') protege todo lo encerrado entre dos comillas simples. Se utiliza para desactivar el significado especial de todos los caracteres. |  |

Ejemplos de uso:

echo "$SHELL"

echo "/etc/\*.conf"

echo "Today is $(date)"

echo '$SHELL'

echo '/etc/\*.conf'

echo 'Today is $(date)'

**1.1.7 . Texto de varias líneas**

Otro problema es como escribir un comando que ocupa varias líneas. Bash nos permite utilizar el carácter de escape para ignorar los cambios de línea de la siguiente forma:

$ echo En un lugar de la Mancha de cuyo nombre no \

> quiero acordarme no a mucho tiempo vivía un \

> hidalgo caballero.

En un lugar de la Mancha de cuyo nombre no quiero acordarme no a mucho tiempo vivía un hidalgo caballero.

Al pulsar intro Bash nos devuelve el segundo prompt, que por defecto es >, y nos permite seguir escribiendo.

Una segunda opción es entrecomillar y no cerrar las comillas, con lo que Bash nos pide que sigamos escribiendo sin necesidad de poner un carácter de escape a los cambios de línea:

$ echo 'En un lugar de la Mancha de cuyo nombre no

> quiero acordarme no a mucho tiempo vivía un

> hidalgo caballero.'

En un lugar de la Mancha de cuyo nombre no quiero acordar no a mucho tiempo vivía un hidalgo caballero.

La diferencia está en que en el primer caso los cambios de línea son escapados, y en el segundo caso no.

**1.2** Escribiendo un Script Shell

Los scripts de shell brindan una forma conveniente de realizar una amplia gama de tareas: recopilar datos, buscar una palabra o frase en un texto y otras cosas útiles. El siguiente ejemplo muestra un pequeño script de shell que imprime un texto:

**EXAMPLE 1.1:**A SHELL SCRIPT que muestra un texto

#!/bin/sh **1**

# Output the following line: **2**

echo "Hello World" **3**

|  |  |
| --- | --- |
| [**1**](https://documentation.suse.com/sles/15-SP1/html/SLES-all/cha-adm-shell.html#co-adm-shell-shebang) | La primera línea comienza con los caracteres Shebang (#!) que indican que este archivo es un script. El intérprete, especificado después de Shebang, ejecuta el script. En este caso, el intérprete especificado es /bin/sh. |
| [**2**](https://documentation.suse.com/sles/15-SP1/html/SLES-all/cha-adm-shell.html#co-adm-shell-comment) | La segunda línea es un comentario que comienza con el signo almohadilla. Le recomendamos que comente las líneas difíciles. Con los comentarios adecuados, puede recordar el propósito y la función de la línea. Además, es de esperar que otros lectores entiendan su guión. Comentar se considera una buena práctica en la comunidad de desarrollo. |
| [**3**](https://documentation.suse.com/sles/15-SP1/html/SLES-all/cha-adm-shell.html#co-adm-shell-echo) | La tercera línea usa el comando incorporado echo para imprimir el texto correspondiente por la salida predeterminada (en esta caso la pantalla) |

Antes de que pueda ejecutar este script, hay algunos requisitos previos:

1. Cada script debe contener una línea Shebang (como en el ejemplo anterior). Si falta la línea, debe llamar al intérprete manualmente (con sh).
2. Puedes guardar el script donde quieras. Sin embargo, es una buena idea guardarlo en un directorio donde el shell pueda encontrarlo. La ruta de búsqueda en un shell está determinada por la variable de entorno PATH. Por lo general, un usuario normal no tiene acceso de escritura a /usr/bin. Por lo tanto, se recomienda guardar sus scripts en el directorio de usuarios ~/bin/. El ejemplo anterior recibe el nombre hello.sh.
3. El script necesita permisos de ejecución. Establezca los permisos con el siguiente comando:

$ chmod +x ~/bin/hello.sh

1. Si ha cumplido con todos los requisitos previos anteriores, puede ejecutar el script de las siguientes maneras:
   1. Como Camino Absoluto. El script se puede ejecutar con una ruta absoluta. En nuestro caso, es ~/bin/hello.sh. Otra manera de ejecutar sería indicando al sistema que el script se encuentra en la carpeta en la que nos hallamos ./hello.sh
   2. En todas partes. Si la variable de entorno PATH contiene el directorio donde se encuentra el script, puede ejecutar el script con hello.sh.

**1.3 Redirigiendo los eventos de ejecución de un comando:**

Cada comando puede usar tres canales, ya sea para entrada o salida:

• **Salida estándar**. Este es el canal de salida predeterminado. Cada vez que un comando imprime algo, utiliza el canal de salida estándar.

• **Entrada estándar**. Si un comando necesita información de los usuarios u otros comandos, utiliza este canal.

•**Error estándar.** Los comandos utilizan este canal para informar de errores.

Para redirigir estos canales, existen las siguientes posibilidades:

Comando > Archivo

Guarda la salida del comando en un archivo, se eliminará el contenido del archivo si este exise. Por ejemplo, el comando ls escribe su salida en el archivo listado.txt

:

$ ls > listing.txt

Comando >> Archivo

Agrega la salida del comando a un archivo. Por ejemplo, el comando ls agrega su salida al archivo listing.txt::

$ ls >> listing.txt

Comando < Archivo

Lee el archivo como entrada para el comando dado. Por ejemplo, el comando de lectura lee el contenido del archivo en la variable:

$ read a < foo

Command1 | Command2

Redirige la salida del comando izquierdo como entrada para el comando derecho. Por ejemplo, el comando cat genera el contenido del archivo /proc/cpuinfo. Esta salida es utilizada por grep para filtrar solo aquellas líneas que contienen cpu:

$ cat /proc/cpuinfo | grep cpu

Cuando ejecutamos un comando, podemos redirigir la entrada, la salida y los errores si se producen cambiando las entradas y salidas standard. Cada canal tiene un descriptor de archivo: 0 (cero) para entrada estándar, 1 para salida estándar y 2 para error estándar. Está permitido insertar este descriptor de archivo antes de un carácter < o >. Por ejemplo, la siguiente línea busca un archivo que comience con foo, pero suprime sus errores redirigiéndolo a /dev/null (don desaparecedrán):

$ find / -name "foo\*" 2>/dev/null

**1.4** Usando Alias

Un alias es una definición de acceso directo de uno o más comandos. La sintaxis de un alias es:

alias *NAME*=*DEFINITION*

Por ejemplo, la siguiente línea define un alias lt que genera una lista larga (opción -l), la ordena por hora de modificación (-t) y la imprime en orden inverso (-r):

$ alias lt='ls -ltr'

Para ver todas las definiciones de alias, utilice alias. Elimine su alias con unaalias y el nombre de alias correspondiente.

**1.5** Usando Variables en Bash

Una variable de shell puede ser global o local. Se puede acceder a las variables globales, o variables de entorno, en todos los shells. Por el contrario, las variables locales solo son visibles en el shell actual.

Para ver todas las variables de entorno, utilice el comando printenv. Si necesita saber el valor de una variable, inserte el nombre de su variable como argumento:

$ printenv PATH

Una variable, ya sea global o local, también se puede ver con echo:

$ echo $PATH

Para configurar una variable local, use un nombre de variable seguido del signo igual, seguido del valor:

$ PROJECT="SLED"

No inserte espacios alrededor del signo igual, de lo contrario obtendrá un error. Para establecer una variable de entorno, use exportar:

$ export NAME="tux"

Para eliminar una variable, utilice unset:

$ unset NAME

La siguiente tabla contiene algunas variables de entorno comunes que se pueden usar en sus scripts de shell:

**TABLE 1.5:**VARIABLES ÚTILES DEL ENTORNO

|  |  |
| --- | --- |
| HOME | el directorio de inicio del usuario actual. |
| HOST | El nombre actual del host |
| LANG | when a tool is localized, cuando se localiza idiomáticamente una herramienta, utiliza el idioma de esta variable de entorno. |
| PATH | la ruta de búsqueda del shell, una lista de directorios separados por dos puntos |
| PS1 | especifica el prompt normal impreso antes de cada comando |
| PS2 | especifica el mensaje secundario impreso cuando ejecuta un comando de varias líneas. |
| PWD | Directorio actual de trabajo. |
| USER | Usuario actual. |

**1.5.1** Uso de variables de argumento

Por ejemplo, si tiene el script foo.sh, puede ejecutarlo así::

& foo.sh "Tux Penguin" 2000

Para acceder a todos los argumentos que se pasan a su secuencia de comandos, necesita parámetros posicionales. Estos son $1 para el primer argumento, $2 para el segundo y así sucesivamente. Puede tener hasta nueve parámetros. Para obtener el nombre del script, use $0.

El siguiente script foo.sh imprime todos los argumentos del 1 al 4:

#!/bin/sh´

echo “El nombre del script es: “$0

echo \"$1\" \"$2\" \"$3\" \"$4\"

Si ejecuta este script con los argumentos anteriores, obtendrá:

"Tux Penguin" "2000" "" ""

Prueba:

Haz la prueba creando foo.sh con el echo mencionado y probando si funcionará al ejecutarlo con los parámetros dados.

**1.5.2** Usando Sustitución de Variable:

Las sustituciones de variables aplican un patrón al contenido de una variable, ya sea desde el lado izquierdo o derecho. La siguiente lista contiene las posibles formas de sintaxis:

${VAR#pattern}

elimina la coincidencia más corta posible desde la izquierda de “/”:

$ file=/home/tux/book/book.tar.bz2

$ echo ${file#\*/}

home/tux/book/book.tar.bz2

${VAR##pattern}

elimina la coincidencia más larga posible de la izquierda de “/”:

$ file=/home/tux/book/book.tar.bz2

$ echo ${file##\*/}

book.tar.bz2

${VAR%pattern}

elimina la coincidencia más corta posible de la derecha de “.”:

$ file=/home/tux/book/book.tar.bz2

$ echo ${file%.\*}

/home/tux/book/book.tar

${VAR%%pattern}

elimina la coincidencia más larga posible de la derecha de “.”:

$ file=/home/tux/book/book.tar.bz2

$ echo ${file%%.\*}

/home/tux/book/book

COPY

${VAR/pattern\_1/pattern\_2}

sustituye el contenido de la variable, cambiando PATRON\_1 por PATRON\_2:

$ file=/home/tux/book/book.tar.bz2

$ echo ${file/tux/wilber}

/home/wilber/book/book.tar.bz2

COPY

**1.6** Agrupando y Combinando Comandos:

Los shells le permiten concatenar y agrupar comandos para ejecución condicional. Cada comando devuelve un código de salida que determina el éxito o fracaso de su operación. Si es 0 (cero), el comando fue exitoso, todo lo demás marca un error que es específico del comando.

La siguiente lista muestra cómo se pueden agrupar los comandos:

**Comando1 ; Comando2**

Ejecuta los comandos en orden secuencial. El código de salida no está marcado. La siguiente línea muestra el contenido del archivo con cat y luego imprime sus propiedades de archivo con ls independientemente de sus códigos de salida

> cat filelist.txt ; ls -l filelist.txt

Comando1 && Comando2

Ejecuta el comando derecho, si el comando izquierdo fue exitoso (AND lógico). La siguiente línea muestra el contenido del archivo e imprime sus propiedades de archivo solo cuando el comando anterior fue exitoso (compárelo con la entrada anterior en esta lista):

> cat filelist.txt && ls -l filelist.txt

Comando1 || Comando2

Ejecuta el comando derecho, cuando el comando izquierdo ha fallado (OR lógico). La siguiente línea crea solo un directorio en /home/wilber/bar cuando falla la creación del directorio en /home/tux/foo::

> mkdir /home/tux/foo || mkdir /home/wilber/bar

funcname(){ ... }

crea una función de shell. Puede utilizar los parámetros posicionales para acceder a sus argumentos. La siguiente línea define la función hola para imprimir un mensaje corto:

> hello() { echo "Hello $1"; }

Puedes llamar a esta función así:

> hello Tux

Lo que imprime:

Hello Tux

**1.7** Trabajar con construcciones de flujo comunes (if – for- While - Until)

Para controlar el flujo de su secuencia de comandos, un shell tiene construcciones while, if, for y case.

**1.7.1** El comando de control if

El comando if se usa para verificar expresiones. Por ejemplo, el siguiente código comprueba si el usuario actual es Tux::

if test $USER = "tux"; then

echo "Hello Tux."

else

echo "You are not Tux."

fi

La expresión de prueba puede ser tan compleja o simple como sea posible. La siguiente expresión comprueba si el archivo foo.txt existe::

if test -e /tmp/foo.txt ; then

echo "Found foo.txt"

fi

La expresión de prueba también se puede abreviar entre corchetes::

if [ -e /tmp/foo.txt ] ; then

echo "Found foo.txt"

fi

Ejemplo de script con IF (para practicar):

Escribe el script en tu ordenador, dentro de tu HOME y en una carpeta llamada Scripts y ejecútalo. El comando read con la opción –p lee desde el prompt. La letra n se convierte a través del read en parámetro. “test” es un comando que compara valores (entre otras cosas), -ge es un parámetro que compara si es mayor (g) o igual (e) que 0.

#!/bin/bash

read -p "Introduce un número positivo o negativo : " n

if test $n -ge 0

then

echo "$n es positivo."

else

echo "$n es negativo."

Fi

Otra manera de escribir el script seria poniendo test en su forma reducida:

#!/bin/bash

read -p "Introduce un número positivo o negativo : " n

if [ $n -ge 0 ]

then

echo "$n es positivo."

else

echo "$n es negativo."

Fi

**Nota**: observa el espacio que hay entre los corchetes y el contenido que está entre los mismos.

Otra manera de introducir datos desde consola podría ser usando echo y read como hicimos en los ejemplos del otro día:

#!/bin/bash

echo “Introduce un número positivo o negativo : "

read n

if [ $n -ge 0 ]

then

echo "$n es positivo."

else

echo "$n es negativo."

Fi

**1.7.2** Creando Bucles Con el Comando **for**

El bucle for le permite ejecutar comandos a una lista de entradas. Por ejemplo, el siguiente código imprime información sobre los archivos PNG en el directorio actual:

for i in \*.png; do

ls -l $i

done

**Ejemplos de bucle for**

1. El siguiente script se llamará testdebucle.sh créalo y pruébalo en ~/scripts

*#!/bin/bash*

*for i in 1 2 3 4 5*

*do*

*echo "Welcome $i times."*

*done*

*#Otra manera de escribir el bucle sería:*

*#!/bin/bash*

*for i in {1..5}*

*do*

*echo "Welcome $i times."*

*done*

*#Otra manera sería:*

*#!/bin/bash*

*for (( i=1; i<=5; i++ ))*

*do*

*echo "Welcome $i times."*

*done*

**Guárdalo y recuerda:**

$ chmod +x testforloop.sh

$ ./testforloop.sh

1. El siguiente script sirve para ejecutar comandos:

*#!/bin/bash*

*# A simple shell script to run commands*

**for** command **in** date pwd df

**do**

echo

echo "\*\*\* The output of $command command >"

*#run command*

$command

echo

**done**

Otro ejemplo que para un bucle ante una coincidencia:

*for ciudad in Manila Bangkok Yakarta Kuala Lumpur*

*do*

*if [[ "$ciudad" == 'Yakarta' ]]; then*

*break*

*fi*

*echo "ciudad: $ciudad"*

*done*

*echo 'Sí, ¡eso es todo!'*

Prueba este bucle e intenta averiguar qué está pasando:

*for i in {1..5}*

*do*

*if [[ "$i" == '4' ]]*

*then*

*continue*

*fi*

*echo "Hai $i4"*

*done*

**1.7.3** Creando Bucles Con el Comando w**hile**

El bucle while en Bash es una estructura de control de flujo que permite ejecutar un conjunto de instrucciones mientras se cumpla una condición determinada. La sintaxis general del bucle while en Bash es la siguiente:

bash

while condición

do

conjunto de instrucciones

done

En esta estructura, la condición se evalúa antes de ejecutar el conjunto de instrucciones dentro del bucle. Si la condición es verdadera (es decir, su valor de salida es 0), se ejecutan las instrucciones dentro del bucle y luego se vuelve a evaluar la condición. Este proceso se repite hasta que la condición sea falsa (es decir, su valor de salida no es 0), momento en el que el bucle se detiene y se continúa ejecutando el resto del script.

Algunos ejemplos de uso común del bucle while en Bash incluyen la lectura de datos de entrada de un usuario, la manipulación de archivos o la iteración sobre una lista de elementos.

Por ejemplo, el siguiente script Bash utiliza un bucle while para imprimir los números del 1 al 10:

#!/bin/bash

contador=1

while [ $contador -le 10 ]

do

echo $contador

contador=$((contador+1))

done

Otra manera:

#!/bin/bash

contador=1

while (( $contador <= 10 ))

do

echo $contador

contador=$((contador+1))

done

En este ejemplo, la condición del bucle while es [ $contador -le 10 ], lo que significa "mientras el valor de la variable 'contador' sea menor o igual a 10". Dentro del bucle, se utiliza el comando "echo" para imprimir el valor actual de la variable "contador" y luego se incrementa su valor en 1 mediante la expresión "contador=$((contador+1))". Este proceso se repite hasta que la variable "contador" alcanza el valor 11, momento en el que la condición se vuelve falsa y el bucle se detiene.

**1.7.4** Creando Bucles Con el Comando Until

El comando "until" es un comando en Linux que funciona de manera similar al comando "while", con la diferencia de que el bucle se ejecuta mientras la condición sea falsa en lugar de verdadera. La sintaxis general del comando "until" en Linux es la siguiente:

bash

until condición

do

conjunto de instrucciones

done

En esta estructura, el conjunto de instrucciones dentro del bucle se ejecuta repetidamente hasta que la condición especificada sea verdadera. Es decir, el bucle se ejecuta mientras la condición sea falsa, y se detiene cuando la condición se convierte en verdadera. En este sentido, el comando "until" es útil para situaciones en las que queremos ejecutar un conjunto de instrucciones hasta que se cumpla una condición específica.

Algunos ejemplos comunes de uso del comando "until" en Linux incluyen la espera de la finalización de un proceso o la verificación de la existencia de un archivo o directorio antes de continuar con la ejecución del script.

Por ejemplo, el siguiente script Bash utiliza el comando "until" para esperar hasta que se complete un proceso antes de continuar con la ejecución del script:

bash

#!/bin/bash

# Esperar a que el proceso 'foo' termine

until ps aux | grep '[f]oo' > /dev/null

do

sleep 1

done

echo "El proceso 'foo' ha terminado"

En este ejemplo, el comando "ps aux | grep '[f]oo' > /dev/null" se utiliza para verificar si hay algún proceso que contenga la cadena "foo" en su nombre. Si se encuentra un proceso que cumpla esta condición, el valor de salida de este comando será 0, lo que se interpreta como verdadero en el contexto del bucle "until". Si no se encuentra ningún proceso que cumpla esta condición, el valor de salida será diferente de 0, lo que se interpreta como falso en el contexto del bucle "until".

Dentro del bucle, se utiliza el comando "sleep 1" para esperar un segundo antes de volver a comprobar la existencia del proceso. Este proceso se repite hasta que se encuentra un proceso que cumpla la condición o hasta que se alcanza un límite de tiempo específico, momento en el que el bucle se detiene y se ejecuta la siguiente instrucción fuera del bucle (en este caso, el comando "echo" que indica que el proceso "foo" ha terminado).

## ANEXO: Rangos

Algo interesante que habrás podido observar en las estructuras for son los rangos. Un **rango**, lo puedes expresar con un número de inicio y un número de fin. Sin embargo, Bash, es lo suficientemente inteligente, como para interpretar que si el número de inicio, es mayor que el número de fin, lo que tiene que hacer es una cuenta regresiva. Así,

* {1..1000} contará de 1 a 1000
* {1000..1} contará de 1000 a 1

Pero, no solo puedes decir que cuente de uno en uno, también le puedes definir paso. Así

* {1..1000..2} contará de 1 a 1000, pero, de dos en dos.
* {1000..1..-2}, lo mismo que en el caso anterior, pero de forma regresiva.

Por supuesto que además de ponerlo en un bucle for, también lo puedes utilizar directamente en un echo, por ejemplo, echo {1..100..5}