El Shell: Bash

Este módulo es un manual breve sobre el shell Bash

Tabla de Contenidos

1 Introducción

El *shell* bash, disponible a través de la orden **bash**, se desarrolló como parte del proyecto GNU. Está basado en la shell **sh** de Unix, y por lo tanto se puede ver como un superconjunto de éste.

Algunas diferencias que podemos encontrar entre ambos son:

- La posibilidad de recuperar órdenes previas mediante un mecanismo de “historia”.
- Facilidades de edición de la línea de comandos.
- La capacidad de conmutar entre procesos y controlar su avance (control de trabajos).
- Formas más flexibles de sustitución de variables.
- Operadores adicionales, de sintaxis similar a la de C.
- Alias para órdenes de uso frecuente, sin tener que usar guiones.

Varias de las órdenes de Bourne again sh **bash** se comportan igual que sus contrapartes de **sh**. Es por ello que en nuestra descripción de **bash** haremos algunas referencias a **sh** para presentar detalles y ejemplos.

En la siguiente tabla se muestra una breve comparativa entre similitudes y diferencias de los shells más populares:

Tabla 1 Algunas diferencias entre los principales shells

	sh	csh	tcsh	ksh	bash
histórico de órdenes	No	Sí	Sí	Sí	Sí
alias	No	Sí	Sí	Sí	Sí
shell scripts	Sí	Sí	Sí	Sí	Sí
autocompletado de archivos	No	Sí	Sí	Sí	Sí
edición de línea de órdenes	No	No	No	Sí	Sí
control de trabajos	No	Sí	Sí	Sí	Sí

2 Salida rápida

En **bash** se puede interrumpir la mayoría de las órdenes si se presiona **`ctrl-c`**, con lo cual se vuelve al indicador de **bash**. Hay tres maneras de salir de **bash**:

- Teclear exit.
- Teclear ctrl-d. Este método no funciona si ***ignoreeof*** ha sido asignada.

3 Sintaxis de las entradas al shell

Cuando el shell **bash** lee su entrada, ya sea de un terminal o de un archivo, descompone la entrada para formar una secuencia de palabras separadas por espacios o tabuladores. Un salto de línea con `escape`, es decir, un salto de línea precedido por una diagonal invertida, también actúa como separador entre palabras. Pueden usarse los saltos de línea con escape para continuar una orden en la línea siguiente.

bash presenta un símbolo de solicitud cuando espera otra orden. La solicitud por omisión es
'**\u@\h:\w\$**',

donde ‘**\u**’ es el nombre de usuario ‘**\h**’ es el nombre del host, ‘**\w**’ representa el signo **~**. Se puede cambiar si se asigna un valor distinto a la variable **PS1**.

fernando@pc1:~$ **PS1="si amo > "**
si amo > **echo hola **
a todos
hola a todos

NOTA: como veremos más adelante, cuando se asigna valor a una varible en el bash shell, es importante no dejar espacios ni antes ni despues del operador de asignación (=)

Además de usar espacios o tabuladores para terminar palabras, éstas pueden concluir con ciertos caracteres que tienen un significado especial para **bash**: **; & | < > ( )**

Una secuencia de uno o más de estos metacaracteres también es una palabra. Por ejemplo, la línea de órdenes:
$ who|ls –l

equivale a la siguiente línea de órdenes, en la que las palabras se separan con espacios:

$ who | ls -l

Cuando **bash** separa una línea de órdenes en palabras, trata como una unidad las cadenas entrecomilladas con comillas (**"**) o apóstrofos (**'**). Si la cadena entrecomillada está rodeada por separadores, se convierte en una palabra; si está junto a otra palabra, se combina con ella para formar una más grande. Todos los separadores dentro de la cadena entrecomillada se consideran como caracteres ordinarios.

$ echo hola >"a b"X *crea un fichero llamado 'a bX' (observe el espacio)*
`holaX`

Comentarios. La entrada a **bash** puede incluir comentarios, que comienzan con un símbolo '#', y se extienden hasta el final de la línea. **Aunque bash admite comentarios en todo momento, donde resultan más útiles los comentarios es como parte de un guión o *script***, al modo que solemos incluir comentarios en el código fuente de cualquier lenguaje de programación.

Órdenes. Una orden simple consta de un nombre de orden, seguida por sus argumentos. El nombre y los argumentos son palabras simples. El nombre puede hacer referencia a un archivo ejecutable o a una orden intrínseca de **bash**, sea ésta simple o compleja. Los operadores

| && || ; & ( )

sirven para formar combinaciones de órdenes simples, por lo que no pueden ser argumentos de una orden simple si no están entrecomillados.

> **$ echo 1;echo 2**
> 1
> 2
> **$ echo 1\;echo 2**
> 1; echo 2

Es posible combinar órdenes simples para formar una compleja, la cual se denomina sencillamente orden. **bash** espera que cada línea de entrada, considerada junto con sus líneas de continuación, contenga una orden (o el inicio de una orden compleja), a la que trata como una expresión construida en base a los operadores previamente indicados. A continuación se presenta el significado de estos operadores:

- **|**: El operador '**|**' conecta dos órdenes a través de un canal. La salida estándar de la primera orden se convierte en la entrada estándar de la segunda. Un grupo de una o más órdenes conectadas de esta forma se denomina interconexión.
- **|&**: Parecido a '|', excepto que el error estándar y la salida estándar de la primera orden se envían a la entrada estándar de la segunda.

> **$ cat no_existo | wc**
> cat: no es posible abrir no_existo
> 0 0 0
> **$ cat no_existo |& wc**

```
      1     6     35
```

- **||**: Si **o1** y **o2** son órdenes, la combinación 'o1 || o2' ejecuta primero **o1**, y luego ejecuta **o2** si el estado de salida de **o1** es distinto de cero. Un estado de salida distinto de cero generalmente indica un error en la ejecución (una posible fuente de confusión es que un estado de salida de cero de una orden significa verdadero, pero un valor de cero para una expresión significa falso).
- **&&**: El operador '&&' se parece a '||', excepto que prueba la ejecución con éxito en lugar de los errores. Si **o1** y **o2** son órdenes, la combinación 'o1 && o2' ejecuta primero **o1**, y luego **o2** si el estado de salida de **o1** es cero. En otras palabras, **o2** sólo se ejecuta si **o1** tiene éxito.

```
$ echo "int main() {return 0;}" > exito.c
$ echo "int main() {return 1;}" > fracaso.c
$ cc –o exito exito.c
$ cc –o fracaso fracaso.c
$ exito || echo hola
$ fracaso || echo hola
hola
$ exito && echo hola
hola
$ fracaso && echo hola
```

- **;**: Si **o** es una orden, la construcción 'o;' hace que **bash** ejecute **o**, y luego, al terminar su ejecución, continúe con la siguiente orden. Por tanto, el efecto de la secuencia 'o1; o2' es "ejecutar **o1** y después ejecutar **o2**".
- **&**: Si **o** es una orden, la construcción 'o &' hace que **bash** comience la ejecución de **o** como proceso en segundo plano, y prosiga de inmediato con la siguiente orden sin esperar a que termine **o**.
- **(...)**: Los paréntesis indican agrupamiento dentro de una orden. Sintácticamente, un grupo de órdenes entre paréntesis hace las veces de una orden simple. Las órdenes entre paréntesis siempre **se ejecutan directamente en un subshell**.

El orden de precedencia, de mayor a menor, de estos operadores es el siguiente:
1) |
2) || &&
3) ;
4) &

Los operadores de órdenes se aplican en orden de precedencia. Las secuencias de '||' y '&&' se aplican de izquierda a derecha. En el caso de estos dos operadores, una orden en segundo plano encerrada entre paréntesis se considera como algo que ocurrirá de inmediato.

```
$ echo x ; echo x | wc –c
x
2
$ (echo x ; echo x) | wc –c
4
```

4 Ejecución de órdenes

bash, al igual que **sh**, tiene tres métodos para ejecutar órdenes: ejecución directa, ejecución directa en un *subshell*, y ejecución indirecta en un *subshell*:

- En la ***ejecución directa***, **bash** ejecuta cada orden conforme la detecta. Toda modificación que ocasione la orden en el estado de **bash**, como un cambio en el valor de una variable de shell, permanece hasta la terminación de **bash**. Las órdenes intrínsecas de **bash** siempre se ejecutan de forma directa, lo mismo que las órdenes contenidas en un archivo utilizado por la orden **source**.
- En la ***ejecución directa en un subshell***, **bash** crea un subshell en el que se ejecutan las órdenes. El subshell hereda una copia de todas las variables del shell padre. Todo cambio de estado en el subshell, en particular las asignaciones a las

variables de shell, no afectan al shell padre. **bash** ejecuta en un subshell las órdenes entre paréntesis, y los procesos en segundo plano.
- En la ***ejecución indirecta en un subshell***, **bash** crea un subshell igual que lo haría para la ejecución directa, pero llama a la rutina de sistema **exec** para que ejecute la orden en dicho subshell. El subshell hereda las variables de entorno, pero no las locales. **Si la orden cambia alguna de las variables de entorno**, estos cambios sólo afectan a las copias y no a los originales. Las órdenes que no son intrínsecas de **bash**, incluidos los guiones de shell y las órdenes implantadas como programas compilados se gestionan por ejecución indirecta en un subshell y se buscan de la manera descrita más adelante en "Búsqueda de órdenes".

Búsqueda de órdenes. Cuando **bash** detecta una orden que no es intrínseca, ya sea un programa compilado o un guión, lo busca en la secuencia de directorios especificada por la variable de *shell path*. **bash** obtiene el valor inicial de *path* de la variable de entorno **PATH**, que obtiene del entorno en el que se hace la llamada a **bash**. La variable path tiene una sintaxis ligeramente distinta a la de **PATH**, pero, en esencia, el significado es el mismo. La búsqueda de órdenes se hace en cada directorio que aparezca en **PATH**, separando cada directorio por ":". Se puede conseguir que la búsqueda de órdenes contenga siempre el directorio de trabajo actual, esto es en el que nos encontramos ubicados
en ese momento, incluyendo el carácter ".".

bash acelera la búsqueda de órdenes mediante una tabla de dispersión (tabla *hash*). Esta tabla registra los nombres de los archivos ejecutables que existen en los directorios indicados en *path*.
Cuidado: si usted crea un archivo ejecutable, sea un guión o un programa compilado, bash no podrá hallarlo si no le indica que reconstruya la tabla de dispersión; esto se puede hacer con la orden **rehash**.

5 Redireccionamiento

UNIX manipula los dispositivos de entrada, salida, más el de salidad para errores como si se tratara de unos archivos en los que se realizan operaciones de E/S, y como tales tienen asignado un número de descriptor.

- Así la entrada tiene el descriptor 0 (*stdin*),
- la salida el 1 (*stdout*) y
- el de error con el 2 (*stderr*).

Por otra parte **UNIX permite redireccionar cualquier descriptor** para que lo se lea/escriba en él se lea/escriba finalmente en el archivo que deseemos. De este modo, con el redireccionamiento puede lograrse que un programa tome su entrada de un archivo específico, en lugar del terminal, o que envíe su salida a un archivo, en lugar de al terminal. Las construcciones de redireccionamiento de **bash** incluyen las básicas, ya descritas, y algunas adicionales.

- **X < *fichero***: **X** Toma la entrada estándar de *fichero*. Esto es, cuando el proceso **X** haga una operación de entrada estándar, no se leerá del teclado como es usual, sino de *fichero*.
- **X << *palabra***: Toma la entrada estándar del texto que sigue, llamado texto de entrada. La variable *palabra* indica el final de la entrada. Cuando **bash** detecta esta construcción en una orden, lee la entrada hasta encontrar una línea que consista exactamente en *palabra*, o bien el final del archivo de entrada. La entrada leída de esta manera se convierte en la entrada estándar de la orden. No se aplican sustituciones de ningún tipo a *palabra*; todas las líneas de entrada se comparan con *palabra* antes de realizar sustituciones en la línea.

Normalmente **bash** lleva a cabo sustituciones de variable y de órdenes en el texto de entrada.
Considera que '****' es un carácter de entrecomillado cuando precede a uno de los caracteres (**\ $ '**), y las demás ocurrencias de '\' se consideran de forma literal, es decir, no se interpretan. Sin embargo, si palabra contiene algunos de los caracteres (**\ " ``**) **bash** toma el texto de entrada de forma literal, y no realiza ninguna de estas

sustituciones.

X [1|2|...|n] >[¡] *fichero*: Envía la **salida estándar** a *fichero*. Si el archivo existe, se trunca su longitud a cero, con lo que se borra su contenido (el borrado afecta a todos los enlaces con el archivo); si no existe, se crea. Sin embargo, si la variable ***noclobber*** está asignada, se inhibe el borrado en caso de que el fichero existiera, y **bash** envía un mensaje de error. La presencia de '**!**' hace que se ignore el valor de ***noclobber*** y que siempre se borre el archivo si existe.

Precediendo el operador de redirección **>** se puede incluir el número de descriptor del archivo que se quiere redireccionar, incluidos la E/S estandard. Por defecto ese descriptor es ***stdout***, el **1**, que se corresponde con la salida estándar. Un uso frecuente es redireccionar el error estándar con **2 >**.. Se puede redirigir la salida estándar de una orden a *fichero1* y el error estándar a *fichero2* con una línea de órdenes con la sintaxis siguiente:

$ (orden > *fichero1*) 2> *fichero2*

[1|2|...|n] >> [!]*fichero*: Esta forma de redireccionamiento es parecida a la anterior, excepto que la salida estándar se anexa al final de fichero. Si la variable ***noclobber*** está asignada, no se permite la reescritura de fichero, será un un error, a menos que se suprima la prueba de error con '**>!**'.

6 Complemento de órdenes

La mayoría de las versiones de **bash** permiten completar nombres de archivo de forma interactiva. Se aplican las normas que siguen:

- Si al final de una orden se teclea un nombre de archivo seguido por *escape* o bien tabulador, **bash** completará el nombre del archivo si existe un complemento único.

> Por ejemplo, si se teclea
>
> **$ fgrep roc av (Tab)**
>
> y el único archivo del directorio actual que empieza por *av* es *ave*, **bash** cambia la línea de entrada a
>
> **$ fgrep roc ave**

Si la variable *FIGNORE* contiene una lista de sufijos separados por comas, los archivos que tengan esos sufijos no serán candidatos elegibles para el complemento de nombre de archivo, a menos que sean los únicos candidatos. Si no existe un complemento único **bash** completa el nombre del archivo hasta donde puede, y luego lista todos los posibles candidatos. Si el número de candidatos es muy elevado no los lista directamente, primero nos informa del número de candidatos disponibles y nos solicita confirmación para proceder al listado.

- Si se teclea un nombre de archivo seguido por *EOF* **bash** proporciona una lista de todos los complementos posibles del nombre, y luego muestra la entrada para que pueda completar el nombre del archivo.

7 Control de trabajos

Un **trabajo** es un grupo de procesos. En **bash**, ***cada grupo de procesos en segundo plano creado con '&' es un trabajo***.

Así mismo, ***un proceso en primer plano interrumpido por una señal de detención (la cual normalmente se envía tecleando ctrl-z) también es un trabajo***.

De todos los trabajos que se ejecutan en un momento determinado, a lo sumo **uno será el trabajo en primer plano**; los **demás serán trabajos en segundo plano o detenidos**. Un trabajo detenido no se ejecutará hasta que se reanude. Cuando **bash** presenta el

indicador esto quiere decir que no hay ningún trabajo en primer plano; en los demás casos, el trabajo en primer plano es el que se indicó con la última orden. Solamente el trabajo en primer plano puede recibir entradas del terminal, aunque cualquier trabajo, sea de primer o segundo plano, puede enviar sus resultados al terminal. Si varios trabajos producen salidas en paralelo, éstas se mezclan.

Al iniciarse un trabajo en segundo plano **bash** presenta su número de trabajo, y su número de proceso asociado al trabajo,

> por ejemplo,
>
> **[2] 470**
>
> En este ejemplo el número de trabajo es **2**, y el número de proceso es **470**.

Una interconexión en segundo plano que contenga varios procesos presentará un número para cada proceso. Cuando un trabajo termina, o cambia su estado, **bash** notifica al usuario que ha ocurrido este cambio justo antes de presentar el siguiente indicador. Si se asigna la variable ***notify*** se puede lograr que **bash** envíe un aviso de los cambios de estado en el momento en que ocurren, incluso si el trabajo se ejecuta en segundo plano.

Listado y referencias de trabajos. La orden **jobs** lista todos los trabajos actuales. Su salida tiene el siguiente formato:

[1] Running	molino
[2] Done	grep oboe vago
[3] Running	sort > lobo
[4] - Stopped	emacs ahorro.c
[5] + Stopped (tty input)	cobrar

La orden **jobs** es un trabajo en primer plano, y por tanto, no se presenta:

- El **número entre corchetes** es el número de trabajo.
- El trabajo actual será el que se haya detenido o iniciado en segundo plano de forma más reciente; este trabajo se indica con '**+**'. El trabajo anterior era el trabajo actual previo, y se indica con '**-**'.
- La tercera columna indica el estado de cada trabajo. Observe que un trabajo puede estar detenido porque espera entrada del terminal o porque se detuvo de forma explícita.
- La última columna muestra la orden asociada a cada proceso de un trabajo. Si se especifica la opción **-l** al ejecutar jobs, se presentan además los números de proceso.

Varias órdenes esperan referencias a trabajos. Hay varias maneras de indicar estas referencias:

- **%n**: El trabajo número n.
- **%cadena**: El único trabajo cuyo nombre comienza por *cadena*. Si sólo existe un trabajo en segundo plano, *cadena* puede estar vacía, y el símbolo '**%**' basta para hacer referencia al trabajo actual.
- **%?cadena**: El único trabajo cuyo nombre contiene *cadena*.
- **%+**
  %%
  %: El trabajo actual.
- **%-**: El trabajo anterior.

Inicio y detención de trabajos. Hay varias maneras de cambiar el estado de un trabajo:

- Se puede detener un trabajo en primer plano si se le envía la señal **STOP**, con lo cual se convierte en un trabajo en segundo plano, y se cambia su estado a '***Stopped***' (detenido). La manera normal de enviar la señal es teclear el carácter de suspensión (normalmente ***ctrl-z***). Para reiniciar un trabajo detenido en segundo plano, use la orden **bg**.
- Puede pasarse el trabajo ***%trabajo*** en segundo plano a primer plano, y otorgarle el control del terminal, si se teclea '**fg *%trabajo***', o simplemente, '***%trabajo***' ante el

indicador del *shell*.
- Puede hacerse que el trabajo ***%trabajo*** detenido se ejecute en segundo plano si se teclea '**bg *%trabajo***' ante el indicador del shell, siempre que el trabajo detenido no esté esperando una entrada del terminal. Si trata de reiniciar en segundo plano un trabajo que espera entrada del terminal, se convertirá en el trabajo actual, pero seguirá detenido.
- Se puede matar al trabajo ***%trabajo*** tecleando '**kill *%trabajo***' ante el indicador del *shell*, siempre y cuando el trabajo no capture o ignore la señal. Existen otras formas de enviar la señal **kill**.Si añadimos el parametro **-9** (**kill -9**) entonces esa señal no puede ser bloqueada, matando incondicionalmente al proceso.

8 Expansión de nombres de fichero

Se pueden usar **comodines en el nombre de un fichero** para representar una secuencia de caracteres. El proceso de expansión de comodines se denomina expansión de nombres de fichero, o, en términos del shell bash, globalización.

El **shell bash** ofrece dos abreviaturas adicionales, además de los comodines usuales:

- El carácter '**~**' al principio de un nombre de archivo representa el **directorio base** o *home* del usuario que llamó a esta copia de bash. Si el símbolo va seguido por el nombre de otro usuario, representa el directorio base del usuario.
- Una cadena de la forma *a{x1,x2,...,xn}b*, en un nombre de archivo representa la lista de nombres: *ax1b ax2b ... axnb*. Tanto *a* como *b* pueden estar vacías. Sin embargo, una ocurrencia de '{', '}' o '{}' rodeada por espacios en blanco se interpreta de forma literal, y no como abreviatura.

> Por ejemplo:
> **$ cp ~noe/fauna/{gaviotas,gerbos,lemures,cebras}**
>
> que copia cuatro archivos del subdirectorio **fauna** de **noe** en el directorio actual. Sin la notación '**{}**', sería necesario repetir cuatro veces la referencia a **~noe/fauna**.

9 Sustituiciones históricas

Una **sustitución histórica** permite **reutilizar una porción de una orden previa al teclear la orden actual**. Las sustituciones históricas reducen la cantidad, y algunos errores, de tecleo.

Una sustitución histórica generalmente comienza con '**!**', y tiene tres partes:

- un *suceso*, que especifica una orden previa,
- un *selector*, que selecciona una o más palabras del suceso, y
- algunos *modificadores*, que alteran las palabras seleccionadas.

El selector y los modificadores son opcionales.

Una sustitución histórica tiene la forma:
![suceso] [[:]selector[:modificador] ...]

El **suceso es obligatorio**, a menos que vaya seguido por un selector que no comience con un dígito. Se puede omitir el símbolo ':' antes del selector si éste no comienza con un dígito. Es posible cambiar el caracter '!' que se usa en las sustituciones históricas por otro si se modifica el valor de *histchars*.

bash interpreta las sustituciones históricas antes de hacer otra cosa, incluso antes de interpretar el entrecomillado y las sustituciones de órdenes. La única manera de entrecomillar el '!' de una sustitución histórica es anteponiéndole una diagonal invertida. Sin embargo, no es necesario entrecomillar el '!' si le sigue un espacio en blanco, '=' o '('.

9.1 Los sucesos y sus especificaciones

bash almacena en una lista histórica las órdenes tecleadas, siempre que consten de un mínimo de una palabra. Las órdenes de la lista histórica se denominan sucesos. Estos sucesos se numeran, asignando el número uno a la primera orden que se emite al entrar a **bash**. En el caso de órdenes complejas, como **foreach**, que constan de más de una línea, sólo se incorpora la primera línea a la lista histórica. La variable *history* especifica cuántos sucesos se conservan en la lista, que puede verse con la orden **history**.

Las formas de especificar un suceso en la sustitución histórica son:

- **!!**: El suceso anterior. La forma más fácil de repetir la orden anterior es teclear '!!'.
- **!n**: El suceso número n.
- **!-n**: El *n*-ésimo suceso anterior. Por ejemplo, '!-1' se refiere al suceso anterior y equivale a '!!'.
- **!cadena**: El único suceso anterior cuyo nombre comienza por *cadena*.
- **!?cadena?**: El único suceso anterior que contiene *cadena*. Puede omitirse el signo '?' final si se coloca un salto de línea.

9.2 Selectores

Se puede seleccionar un subconjunto de palabras de un suceso añadiendo un selector al suceso. Una sustitución histórica sin selector incluye todas las palabras del suceso.

Los selectores que se puede usar para seleccionar palabras de un suceso son:

- **:0** El nombre de la orden
- **[:]^** El primer argumento
- **[:]$** El último argumento
- **:n** El *n*-ésimo argumento (n >= 1)
- **:n1-n2** Las palabras *n1* a *n2*
- **[:]*** Las palabras 1 a $
- **:x*** Las palabras *x* a $
- **:X*** Las palabras *x* a ($ - 1)
- **[:]-x** Las palabras 0 a *x*
- **[:]%** La palabra que coincida con la búsqueda '?cadena?' precedente.

Los dos puntos antes de un selector se pueden omitir si el selector no comienza con un dígito.

9.3 Modificadores

Se puede modificar las palabras de un suceso agregando uno o más modificadores. Cada modificador debe ir precedido por un signo de dos puntos.

Los modificadores que siguen suponen que la primera palabra seleccionada es un nombre de archivo:

- **:r** Elimina el componente '.cadena' final de la primera palabra seleccionada.
- **:h** Elimina el componente final de nombre de trayectoria de la primera palabra seleccionada.
- **:t** Elimina todos los componentes iniciales de nombre de trayectoria de la primera palabra seleccionada.

> Por ejemplo:
>
> **$ ls -l /usr/elsa/juguetes.texto**
> **$ echo !!^:r !!^:h !!^:t !!^:t:r**
> */usr/elsa/juguetes /usr/elsa/ juguetes.texto juguetes*

Los modificadores que siguen permiten hacer sustituciones en las palabras seleccionadas de un suceso. Si el modificador incluye ***g***, la sustitución se aplica a todo el suceso; de lo contrario, sólo se aplica a la primera palabra modificable.

- **:[g]s/l/r** Sustituye la cadena **r** por la cadena **l**. El delimitador '/' se puede reemplazar con otro carácter delimitador. Dentro de la sustitución, el delimitador se puede entrecomillar anteponiéndole '\' . Si l está vacía, se emplea la cadena más reciente que se haya usado, ya sea una cadena l previa o la cadena de un selector de suceso de la forma '!?cadena?'. El delimitador final se puede omitir si se emplea un salto de línea.
- **:[g]&** Repite la sustitución previa.

Los modificadores que siguen entrecomillan las palabras seleccionadas, posiblemente después de otras sustituciones:

- **:q** Entrecomilla las palabras seleccionadas, evitando sustituciones posteriores.
- **:x** Entrecomilla las palabras seleccionadas pero divide el texto en palabras, basándose en el espacio en blanco.

El modificador que sigue permite ver el resultado de una sustitución sin ejecutar la orden resultante:

- **:p** Muestra la nueva orden, pero no la ejecuta.

9.4 Convencionalismos especiales

Los convencionalismos especiales que siguen ofrecen abreviaturas para las formas de sustitución histórica de uso más frecuente:

- Una especificación de suceso se puede omitir de la sustitución histórica si va seguida por un selector que no comienza con un dígito. En este caso se considera que el suceso es el que se usó en la referencia histórica más reciente de la misma línea, si la hubo, o bien el suceso previo, en otro caso.

  > Por ejemplo, la orden:
  > **$ echo !?quetzal?^ !$**
  >
  > presenta el primer y último argumento de la orden más reciente que contenga la cadena *quetzal*.

- Si el primer carácter distinto de espacio en blanco de una línea de entrada es '^', este símbolo se considera como abreviatura de '!:s^'. Esta forma facilita la corrección de un error de tecleo en la línea anterior.

  > Por ejemplo, si por error tecleó la orden
  >
  > **$ cat /etc/lasswd**
  >
  > puede volver a ejecutarlo, cambiando 'lasswd' a 'passwd', si teclea
  > ^l^p

  Se puede reemplazar '^' con un carácter distinto si se modifica el valor de *histchars*.

- Se puede encerrar una sustitución histórica entre llaves para evitar que absorba los caracteres que siguen. En este caso, toda la sustitución, con excepción del '!' inicial, deberá estar entre las llaves.

  > Por ejemplo, suponga que emitió la orden
  >
  > **$ cp cuenta ../dinero**
  >
  > Entonces, '!eps' buscará una orden anterior que comience con 'eps', pero '!{cp}s' se convertirá en la orden

> **$ cp cuentas ../dineros**

10 Alias

Se puede definir un alias como la abreviatura de una orden de uso frecuente. Los alias se crean, presentan y eliminan con las órdenes **alias** y **unalias**. **bash** mantiene una lista de los alias; cada alias de la lista relaciona una palabra de orden con el texto que lo define.

> Algunos ejemplos sencillos:
>
> **alias h=history**
> **alias ll=ls -l**

Cuando **bash** lee una línea de órdenes, determina si la primera palabra de la línea es un alias. De ser así, **bash** reemplaza la palabra con la definición del alias, de la siguiente manera:

- Si la definición no contiene sustituciones históricas, el nombre de orden se reemplaza directamente con su definición.
- Si la definición contiene sustituciones históricas, las sustituciones se aplican como si el texto de la orden fuera la línea de entrada previa. El resultado reemplaza toda la línea de órdenes. Estas sustituciones históricas permiten que la definición del alias utilice los argumentos de la línea de órdenes.

Si lo requiere, puede negar un alias colocando antes una diagonal invertida, con lo que la palabra del alias tendrá su significado original.

Si una sustitución de alias no cambia la primera palabra de la línea de órdenes, la sustitución llega a su fin. De lo contrario, **bash** lleva a cabo las demás sustituciones hasta que ya no se aplique ninguna o detecte un ciclo.

Una sustitución de alias como

> **$ alias who='who;date'**

es válida: el resultado de la sustitución no altera la primera palabra, de manera que **bash** ya no la sigue procesando. Las comillas simples previenen al shell para que no intente interpretar el texto entrecomillado, en particular el carácter “;”, que no queremos que marque el final de la sentencia alias, sino que sea parte de la asignación.

Sin esas comillas,

$ alias who=who;date

se intepretaria como:

$ alias who=who

$ date

Un alias de '*shell*' recibe un tratamiento especial: **bash** usa su valor como el *shell* que deberá llamar cuando ejecute una orden en un *subshell*. La definición del alias debe consistir en el nombre de la trayectoria completa del *shell*.

> **Ejemplos.** A continuación se presentan algunos ejemplos de la aplicación de los alias:
>
> **$ alias ls ls –CF**
>
> la orden 'ls subdir' se transformará en 'ls -CF subdir', pero '\ls subdir' se ejecutará sin modificaciones.
>
> **$ alias buscar 'who | fgrep \!^'**
>
> 'buscar igor' se transformará en 'who | fgrep igor', que presenta los terminales donde está conectado 'igor'.

11 Variables de shell

bash mantiene un conjunto de variables de *shell*, cada variable contiene una lista de cero o más palabras. Las variables de shell incluyen **variables predefinidas** y **variables definidas por el usuario**. Se puede recuperar los valores de estas variables y asignarles nuevos valores. También se puede recuperar o asignar las palabras individuales dentro de una variable. Una variable de *shell* es un tipo especial de variable local. La diferencia entre una variable de entorno y una local o de *shell*, es similar a la que encontramos entre una variable global y local. Una **variable de entorno** una vez fijada es accesible por cualquier proceso que se lance, es un atributo del sistema operativo, mientras que una **local sólo es conocida desde la misma shell que la creó**.

El shell bash permite además recuperar y asignar variables de entorno. El shell bashhace una distinción explícita entre las variables de entorno y las de shell, y cuenta con órdenes distintas que seaplican a cada tipo:

- El operador de asignación “=”asignan variables de *shell*.
- La orden **unset** elimina tanto variables de shell como de entorno.
- La orden **export** asigna y presenta variables de entorno.
- La orden **export** presenta las variables de entorno.
- La orden **unsetenv** elimina variables de entorno.

Si una variable de shell y una de entorno tienen el mismo nombre, la variable de shell tiene precedencia sobre la de entorno en todas las sustituciones de variables.

12 Sustitución de variables

Una sustitución de variable se refiere al valor de una variable de shell, un parámetro, o una de varias cadenas que controla bash. La sustitución produce texto que se sustituye en la línea de órdenes. En estas construcciones se pueden omitir las llaves si no hay ambigüedad. Todas las construcciones que producen una secuencia de palabras se pueden alterar con los modificadores que se presentan en "Modificadores para la sustitución de variables”.

Las sustituciones de variables comienzan con '$'. Si desea que un '$' se interprete de forma literal, deberá escribirlo como '\$'. Sin embargo, esto no es necesario si el '$' va seguido por un espacio en blanco.

12.1 Variables con nombre

Las sustituciones de variables que se presentan a continuación atañen a las variables con nombre:

Las sustituciones de variables que se presentan a continuación atañen a las variables con nombre:

${nombre} o simplemente $nombre: Produce las palabras de la variable nombre.

La primera palabra ocupa la posición ordinal 1. Los selectores pueden ser el resultado de una sustitución de variables.

${#nombre}: Produce el número de caracteres de la variable nombre, con formato numérico decimal.

> Ejemplos:
>
> **$ x=3** *Se crea la variable x con valor 3*
> **$ echo $x ${x}** *Se muestra 2 veces el valor de x*
> 3 3
> **$ echo $xp**

> `xp: Undefined variable` *No existe una variable llamada xp*
> **$ echo ${x}p** *Con las llaves indicamos que la sustitución solo atañe a x*
> `3p`

12.2 Otras sustituciones de variables

- **$#** Es remplazado por la cantidad de parámetros que el ***script*** recibe.
- **$*** Que se expande a todos los parámetros que el ***script*** haya recibido, un parámetro se separa de otro con el valor de la variable IFS que normalmente es un espacio.
- **$?** Todo programa al terminar debe retornar un número al sistema operativo, por convención 0 significa operación exitosa y números diferente representan errores. $? se expande al número retornado por el último programa ejecutado en primer plano. Un script puede retornar un 3 en lugar de 0 con exit 3
- **$-** Opciones que se pasaron al script durante su ejecución.
- **$$** Identificación del proceso del intérprete de comandos.
- **$!** Identificación del proceso del último comando que se ejecutó en segundo plano.
- **$0** Nombre del script o del shell.
- **$1-$9** Devuelve del primer al noveno argumento con los argumentos que fue invocado el ***script***.

13 Entrecomillado

El entrecomillado puede servir para evitar que los metacaracteres, incluido el espacio en blanco, tengan su interpretación normal. Las formas de entrecomillado que reconoce **bash** son:

- **\c**: Entrecomilla el carácter **c**. Esta forma de entrecomillado es la única que se puede usar para el carácter de sustitución histórica (generalmente '!').
- **"texto"**: Entrecomilla ***texto***, pero se siguen reconociendo como metacaracteres ($ ` !) dentro de *texto*. Por consiguiente, se respetan las sustituciones de variables, de órdenes e históricas. Una diagonal invertida antes de un signo de admiración o de un salto de línea entrecomilla el símbolo; en cualquier otro contexto se considera como un carácter normal, incluso si está antes de otra diagonal invertida. Como el espacio en blanco se interpreta literalmente todo el texto entrecomillado actúa como una sóla palabra. La única excepción es que un salto de línea producido por una sustitución de órdenes actúa como separador entre palabras.
- **'texto'**: Igual que "texto", pero sólo se reconoce como metacarácter el símbolo '!' en el texto. Como sucede con "texto" los signos de admiración y los saltos de línea son los únicos caracteres que se pueden entrecomillar con una diagonal invertida.

> **$ perro=pluto**
> **$ echo \! "[$perro; `echo v`;]"**
> **'[\$terrier; `echo vera`;]'**
> ! [pluto; v;] [$pluto; `echo v`;]

y cada porción de esta salida que esté entre corchetes se considerará una sola palabra, a pesar de los espacios en blanco.

14 Sustitución de órdenes

La sustitución de órdenes puede servir para ejecutar una orden y usar inmediatamente su salida como parte de otra.

`texto` (esta comilla es la inversa, se llama de ejecución) Ejecuta *texto* indirectamente como una orden en un subshell, y luego reemplaza *texto* en la línea de órdenes con la salida estándar de la ejecución. Si el resultado termina con un salto de línea, éste se descarta. El espacio en blanco en la salida separa las palabras de la línea de órdenes resultante, aunque no haya ningún espacio en blanco implícito al inicio ni al final de la sustitución de órdenes.

Si la sustitución ocurre en un entrecomillado que usa comillas, los espacios y tabuladores en la salida no actúan como separadores de palabras, pero sí los saltos de línea. Estos últimos representan la única manera en que el material encerrado entre comillas puede contener más de una palabra.

$v1=(X`echo a b c`Y)
$$ echo $vl Xa b cY

15 Expresiones

Varias órdenes intrínsecas de **bash** aceptan expresiones que tienen casi la misma forma que las expresiones de C. Los operandos de estas expresiones son números decimales, números octales, cadenas o pruebas. Una secuencia de dígitos que comienza por cero se considera como un número octal. Las pruebas se describen en "Pruebas". Los operandos se pueden obtener como resultado de sustituciones de variables.

Los operadores, en orden descendente de precedencia, que reconoce bash son:

- **(...)** agrupamiento
- para comparar enteros:

  [integer1 -eq integer2]

  - -eq igual (equal).
  - -ne distinto (not equal).
  - -ge mayor o igual (greater than or equal).
  - -gt mayor (greater than).
  - -le menor o igual (less than or equal).
  - -lt menor (less than).
- para comparar o aplicar a cadenas:

  [string1 = string2], [-z string1]

  - = igual.
  - != distinto.
  - -z la longitud de la cadena es cero.
  - –n la longitud de la cadena es mayor que cero.
  - < es menor que
  - > es mayor que
- para comparar o aplicar a archivos:

  [file1 -nt file2] , [-e file1]

  - -nt fecha de modificación más reciente (newer than).
  - -ot fecha de modificación más antigua (old than).
  - -e el archivo existe.
  - -s el archivo existe y tiene un tamaño mayor que cero.
  - -d el archivo existe y es un directorio.
  - -h el archivo existe y es un enlace simbólico.
  - -x el archivo existe y es ejecutable.
- operadores lógicos:

  - –o OR
  - –a AND
  - -¡ NOT

16 Variables predefinidas

Las variables que se presentan a continuación tienen significado predefinido para **bash**. Algunas tienen el mismo significado en sh.

Directorios y ficheros

Las variables que siguen indican dónde están ciertos directorios y ficheros:

- **HOME**: El directorio base del usuario que llamó al shell. Su valor inicial es el valor de la variable de entorno HOME.
- **PWD**: La trayectoria absoluta del directorio de trabajo del shell. **bash** actualiza esta variable cuando cambia de directorio. Un modo rápido de conocer el valor de nuestro directorio es con el comando *pwd*.
- **PATH**: La lista de directorios para la búsqueda de órdenes. Inicialmente se le asigna el valor de la variable de entorno PATH.
- **SHELL**: El fichero en el que reside el shell. Su valor inicial es la ubicación de **bash**.

Los valores de HOME y PATH en el entorno heredado por **bash** se actualizan cada vez que cambian *home* y *path*, de manera que los programas que se llamen desde **bash** vean los valores actualizados. Sin embargo, los cambios en estas variables de entorno se pierden con la terminación de **bash**.

17 Órdenes intrísecas simples

Las órdenes que se describen a continuación son intrínsecas, y no contienen otras órdenes:

17.1 Alias

- **alias [nombre=[texto]]**: Si se emplea sin opciones, alias muestra todos los alias existentes. Si se usa sólo con *nombre*, muestra el alias que existe con ese nombre. Si se usa con el nombre y algún texto, asigna a *nombre* las palabras de *texto*.
- **unalias pat**: Elimina todos los alias que coincidan con el patrón *pat*. El patrón puede contener los mismos comodines que se emplean en la expansión de nombres de fichero.

17.2 Administración de Directorios

bash ofrece una pila de directorios para que sea más fácil volver a los que se visitaron anteriormente. La pila se puede considerar como una lista; al sacar (*pop*) de la pila se quita el primer elemento de la lista, mientras que al meter (*push*) un directorio en la pila se coloca a inicio de la lista. El primer elemento de la pila tiene el número 0.

Las órdenes que siguen cambian o presentan el directorio actual, o la pila de directorios:

- **cd [dir]**
- **chdir [dir]**: Cambia el directorio de trabajo a *dir*. Si no se especifica *dir*, el cambio es al **directorio base**.
- **dirs**: Muestra la pila de directorios.
- **popd**: Saca un directorio de la pila, y lo convierte en el directorio actual.
- **popd +n**: Descarta el *n-ésimo* elemento de la pila, sin modificar el directorio actual
- **pushd**: Intercambia los dos elementos superiores de la pila de directorios, y después hace que el directorio de la cima de la pila sea el directorio actual. **pushd** sirve para cambiar fácilmente entre dos directorios
- **pushd dir**: Mete el directorio actual en la pila y luego cambia el directorio actual a *dir*.
- **pushd +n**: Lleva el *n-ésimo* elemento de la pila a la parte superior, pasando los elementos anteriores uno por uno al fondo de la pila; después hace que el directorio actual sea el que está en la cima de la pila.

No todas las versiones de **bash** cuentan con las órdenes **dirs**, **popd** y **pushd**.

17.3 Salida

Las órdenes que siguen representan formas de salir de **bash**:

- **logout**: Termina el *shell* si es el de entrada al sistema.
- **login**: Termina el *shell*, si es el de entrada al sistema, y luego solicita una nueva entrada.
- **exit [(expr)]**: sale del *shell* con *expr* como estado de salida. Si se llama a **exit** entre paréntesis, **bash** no termina, porque dicha salida es sólo del *subshell*.
- **exec orden**: Ejecuta orden en lugar del *shell* actual. Este orden es la forma más común de cambiar de *shell*.

17.4 Presentación de argumentos / Información de estado

Las órdenes que siguen **presentan** en la pantalla sus **argumentos**:

- **echo [-n] [palabra] ...**: Presenta palabra ... en la salida estándar. La opción ***-n*** suprime el salto de línea final.

Las órdenes que sigue proporcionan **información de estado** y permiten cambiar ciertos parámetros.

- **history [-r] [-h] [ ]**: Presenta los ***n*** sucesos más recientes de la historia de órdenes. Si se especifica ***-r***, los sucesos se presentan en orden inverso, comenzando por los más recientes. Si especifica ***-h***, se omiten los números iniciales de la lista histórica, esto es adecuado para su redireccionamiento a un fichero y su ejecución posterior como guión. Estas opciones deben estar rodeadas por espacios en blanco.
- **jobs [-l]**: Lista los trabajos activos. El listado incluye el número de proceso de cada trabajo si se especifica la opción **-l**.
- **umask [n]**: Si **n** está presente, lo asigna como valor de la máscara de creación de ficheros; de lo contrario, muestra la máscara.

17.5 Control de trabajos y procesos

Las órdenes que siguen permiten **controlar los procesos y los trabajos**:

- **kill [-señal] id**: Si se especifica *señal*, ésta es enviada al trabajo o proceso identificado mediante ***id***. Un proceso se especifica con su **pid**, y un trabajo con su identificador de trabajo. La señal se puede especificar con su nombre (omitiendo SIG), o con su número. Si se omite *señal* se usa TERM (señal 15).
- **kill –l**: Lista los nombres de todas las señales permitidas.
- **nohup [orden]**: Si se especifica *orden*, ésta se ejecuta ignorando las desconexiones del terminal. De lo contrario, se ejecuta lo que quede del proceso actual ignorando las desconexiones del terminal.
- **nice [+n] [orden]**: Si se especifica una *orden* y *n*, se ejecuta la orden con prioridad *n + 20*. Si se omite *n*, el valor por omisión será *24*. La prioridad se reduce al aumentar el número. Si se omite *orden*, se ejecuta el resto del proceso actual con la prioridad indicada. El propósito de esta orden es ayudar a los demás usuarios, permitiéndoles que sus trabajos tengan mayor prioridad que el actual.
- **wait**: Espera a la terminación de todos los trabajos en segundo plano.
- **trap '' INT** Esta orden hace que se ignore la interrupción INT
- **trap – INT** Esta orden restaura las acciones por omisión de las interrupciones

17.6 Órdenes diversas

- ***hash añade*** a la tabla de búsqueda de nombres de órdenes.
- **hash –r** reinicia la tabla de búsqueda de nombres de órdenes.
- **source [-h] fichero**: Lee las órdenes de fichero y las ejecuta sin crear un *subshell*. Si las órdenes asignan valores a variables locales, se conservan las asignaciones. Las órdenes no se colocan en la lista histórica, a menos que se especifique la opción ***-h***, con lo que se colocan en la lista, pero no se ejecutan.
- **eval arg ...** : Realiza sustituciones textuales en *arg* ..., y luego ejecuta la orden resultante. **eval** se puede usar para construir órdenes que serían difíciles de especificar de otra manera.
- **time [orden]**: Si se especifica *orden* la ejecuta, y muestra el tiempo que utilizó. De lo contrario, indica
cuánto tiempo ha usado el *shell*.
- **dirname** y **basename** permiten, respectivamente extraer el *path* y el *nombre del archivo* de .

18 Órdenes compuestas (enunciados)

Las **órdenes compuestas** que se presentan a continuación, también llamadas **enunciados**, permiten crear pruebas **condicionales**, pruebas **de caso** y **ciclos**.

Una lista de órdenes puede ocupar varias líneas e incluir enunciados anidados. Las partes de una orden compuesta generalmente se deben colocar en líneas separadas, como se muestra en la sintaxis de la orden. Si se ejecuta una orden compuesta de forma interactiva, **bash** solicita las líneas de continuación. En la lista histórica sólo aparece la primera línea de una orden compuesta.

18.1 if

if [expr]; orden: Ejecuta *orden* si *expr* es distinto de cero (verdadero). Nótese que *expr* d***ebe ir entre corchetes***.

```
1 if [ expr ]; orden
```

```
1 if [ expr ]; then
2     listamandatos1
3 [else
4     listamandatosn]
5 fi
```

Se prueba cada *expr* hasta encontrar una con valor distinto de cero (verdadero), para luego ejecutar su lista de órdenes correspondiente, *listamandatosi*. Si no hay ninguna expresión distinta de cero yexiste una parte '**else**', se ejecuta su lista de órdenes.

18.2 case

case: Permite seleccionar las órdenes que se ejecutarán de acuerdo con la coincidencia con una cadena. El formato del enunciado es:

```
1 case expr in
2 cad1)
3      listamandatos ;;
4 cad2)
5      listamandatos ;;
6 *)
7      listamandatos ;;
8 esac
```

El enunciado puede contener cualquier número de casos. *expr* se expande con todas las sustituciones aplicables. Después se examina cada etiqueta de caso. La cadena ***cad*** se trata como un patrón y se compara con cadena. Si es posible obtener cadena a partir de ***cad*** con la expansión de nombres de fichero, es decir, expandiendo los comodines, se selecciona el caso, y se ejecuta su lista de órdenes y las demás listas de órdenes relacionadas con otros casos, hasta llegar a '**esac**' o detectar una orden**;;**. Lo usual es terminar cada caso con **;;** para evitar que el control pase al siguiente caso.

Si ninguno de los casos coincide, y existe un enunciado por defecto (default), entonces se ejecuta su lista de órdenes. Si ninguno de los casos coincide y no hay enunciado por defecto, concluye la ejecución de todo el enunciado.

18.3 for

```
1 for nombre in secuencia
2 do
3     listamandatos
4 done
```

La lista de órdenes se ejecuta una vez por cada palabra de la lista *secuencia*..., asignando a *nombre* el elemento actual. *secuencia* ... se obtiene generalmente como resultado de una expansión de nombres de fichero con comodines, o capturando la salida de la ejecución de algún comando mediate los operadores de sustitución.

Algunos ejemplos:

```
1 for i in 1 2 3 4 5
2 do
3     echo "Hola $i veces"
4 done
```

```
1 for i in $( ls );
2 do
3     echo item: $i
4 done
```

```
1 for i in `ls`;
2 do
3     echo item: $i
4 done
```

18.4 until / while

Otros bucles usuales son los bluces tipo **until** y **while**:

```
1 until [expre];
2 do
3    listamandatos
4 end
```

```
1 while [expr];
2 do
3     listamandatos
4 done
```

Las órdenes de la lista se ejecutan mientras el valor de la expresión *expr* sea distinto de cero (verdadero). La expresión se evalúa antes de cada ejecución de *listamandatos*, incluyendo la primera vez.

Las órdenes de la lista se ejecutan mientras el valor de la expresión *expr* sea distinto de cero (verdadero). La expresión se evalúa antes de cada ejecución de *listamandatos*, incluyendo la primera vez.

19 Iniciación

Cuando **bash** se inicia, busca un fichero llamado *.bashrc* en el directorio base, y ejecuta las órdenes que contiene. Si **bash** fue llamado como shell de entrada, ejecuta el fichero *.login* tras *.bashrc*, y *.logout* cuando termina.

La ejecución de *.login* y *.logout* puede ser útil, pero sólo ocurre cuando bash es el shell de entrada.

Por ejemplo, si **sh** es el shell de entrada, se puede cambiar a **bash** si se coloca al final del fichero *.profile* la orden **exec bash**

En este caso no se ejecutan *.login* ni *.logout*. Sin embargo, es posible engañar a **bash** para que piense que fue llamado como *shell* de entrada si se crea un sinónimo '**-bash**' para **bash** (con la orden **ln**) y la orden **exec** hace referencia a '**-bash**' . **bash** determina si fue llamado como *shell* de entrada al sistema examinando el primer carácter del nombre con el que se le llamó, que debe ser '-' en el caso de un *shell* de entrada al sistema.