Manual de Referencia de Bash

Documentación de Referencia para Bash
Edición 5.1, para `Bash` Versión 5.1.
October 2020

Chet Ramey, Case Western Reserve University
Brian Fox, Free Software Foundation

Este texto es una breve descripción de las funcionalidades presentes en el intérprete de órdenes de Bash (version 5.1, 29 October 2020).

Esta es la Edición 5.1, actualizada por última vez 29 October 2020, de *Manual de Referencia de Bash*, para `Bash`, Version 5.1.

Índice General

1 Introducción

1.1 Qué es Bash?

Bash es el intérprete, o el lenguaje intérprete de órdenes, para el sistema operativo GNU. El nombre es un acrónimo de '`Bourne-Again SHell`', un juego de palabras con Stephen Bourne, el autor del ancestro directo del actual intérprete `sh`, que apareció en la versión Seventh Edition Bell Labs Research de Unix.

Bash es en gran parte compatible con `sh` e incorpora funcionalidades útiles del intérprete Korn `ksh` y el intérprete C `csh`. Esta concebido para ser una implementación que se ajusta a la parte IEEE POSIX Shell and Tools de la especificación IEEE POSIX (IEEE Standard 1003.1). Ofrece mejoras funcionales sobre `sh` tanto para uso interactivo como para programar.

Aunque el sistema operativo GNU proporciona otros intérpretes de órdenes, incluyendo una versión de `csh`, Bash es el intérprete predeterminado. Al igual que otro software de GNU, Bash es bastante portable. Actualmente se ejecuta en casi cualquier versión de Unix y algunos otros sistemas operativos —existen versiones portadas mantenidas de forma independiente para MS-DOS, OS/2 y plataformas Windows—.

1.2 Qué es un intérprete?

Básicamente, un intérprete es simplemente un procesador de macros que ejecuta instrucciones. El término procesador de macros significa funcionalidad donde texto y símbolos son expandidos para crear expresiones más grandes.

Un intérprete de Unix es tanto un intérprete de órdenes como un lenguaje de programación. Como un intérprete de instrucciones, el intérprete proporciona la interfaz de usuario a un variado conjunto de utilidades de GNU. Las funcionalidades del lenguaje de programación permiten que estás utilidades se combinen. Se pueden crear archivos que contienen instrucciones, y convertirse ellos mismos en instrucciones. Estas nuevas órdenes tienen la misma naturaleza que instrucciones del sistema en directorios como `/bin`, permitiendo que usuarios o grupos puedan establecer entornos personalizados para automatizar sus tareas comunes.

Los intérpretes pueden ser usados de forma interactiva o de forma no interactiva. En el modo interactivo, aceptan la entrada escrita desde el teclado. Cuando se ejecutan de forma no interactiva, los intérpretes ejecutan instrucciones leídas de un archivo.

Un intérprete permite la ejecución de instrucciones GNU, tanto síncrona como asíncronamente. El intérprete espera a que las instrucciones síncronas se completen antes de aceptar más entradas; las instrucciones asíncronas continúan ejecutándose en paralelo con el intérprete mientras que lee y ejecuta instrucciones adicionales. Las construcciones de *redirección* permiten un control preciso de la entrada y la salida de esas instrucciones. Asimismo, el intérprete proporciona control sobre los contenidos de los entornos de instrucciones.

Los intérpretes también proporcionan un pequeño conjunto de instrucciones integradas (*builtins*) que implementan funcionalidad imposible o inconveniente de obtener mediante utilidades separadas. Por ejemplo, `cd`, `break`, `continue` y `exec` no pueden ser implementados fuera del intérprete porque manipulan directamente el intérprete en sí mismo. Las instrucciones integradas `history`, `getopts`, `kill` o `pwd`, entre otras, podrían ser implementadas

en utilidades separadas, pero son más convenientes de usar como instrucciones integradas. Todas las funciones integradas del intérprete son descritas en las secciones posteriores.

Mientras que la ejecución de instrucciones es esencial, la mayoría del poder (y de la complejidad) de los intérpretes es debido a sus lenguajes de programación integrados. Como cualquier otro lenguaje de alto nivel, el intérprete proporciona variables, estructuras de control de flujo, entrecomillado y funciones.

Los intérpretes ofrecen funcionalidades enfocadas específicamente para uso interactivo en vez de para aumentar el lenguaje de programación. Estas funcionalidades interactivas incluyen control de tareas, edición en línea de órdenes, historial de instrucciones y aliases. En este manual se describe cada una de estas funcionalidades.

2 Definiciones

Estas definiciones son usadas a lo largo de este manual.

`POSIX` Una familia de estándares de sistemas abiertos basado en Unix. Bash se atañe principalmente a la porción Shell and Utilities del estándar POSIX 1003.1.

`vacío` Un carácter de espacio o tabulación.

`instrucción integrada`
: Una instrucción que está implementada internamente por el propio intérprete, en vez de por un programa ejecutable en algún lugar del archivo de ficheros.

`operador de control`
: Un `símbolo` que realiza una función de control. Es una `nueva línea` o uno de los siguientes: '`||`', '`&&`', '`&`', '`;`', '`;;`', '`;&`', '`;;&`', '`|`', '`|&`', '`(`', o '`)`'.

`estado de salida`
: El valor devuelto por una instrucción a su ejecutor. El valor está restringido a ocho bits, así que el máximo valor es 255.

`campo` Una unidad de texto que es el resultado de una de las expansiones del intérprete. Después de la expansión, cuando se ejecuta un comando, los campos resultantes son usados como el nombre de instrucción y los argumentos.

`nombre de archivo`
: Una cadena de caracteres usada para identificar a un archivo.

`tarea` Un conjunto de procesos que componen una tubería, y cualquier otro proceso que descienda de ella, que se encuentran todos en el mismo grupo de proceso.

`control de tareas`
: Un mecanismo por el cual los usuarios pueden selectivamente parar (suspender) o reiniciar (reanudar) la ejecución de procesos.

`metacarácter`
: Un carácter que, cuando no está entrecomillado, separa palabras. Un metacarácter es `espacio`, `tabulación`, `nueva línea` o uno de los siguientes caracteres: '`|`', '`&`', '`;`', '`(`', '`)`', '`<`', or '`>`'.

`nombre` Una `palabra` que solo está compuesta de letras, números y barras bajas, y comienza por una letra o barra baja. Los `Nombre`s son usados como nombres de variables y funciones del intérprete. También mencionado como `indentificador`.

`operador` Un `operador de control` o un `operador de redirección`. Véase Sección 3.6 [Redirecciones], página 37, para una lista de los operadores de redirección. Los operadores contienen al menos un `metacarácter` sin entrecomillar.

`grupo de proceso`
: Una colección de procesos relacionados que tienen el mismo ID de proceso de grupo.

`ID de proceso de grupo`
: Un identificador único que representa un `proceso de grupo` durante su tiempo de vida.

`palabra reservada`
: Una `palabra` que tiene un significado especial para el intérprete. La mayoría de palabras reservadas introducen construcciones de control de flujo, como `for` y `while`.

`estado de retorno`
: Un sinónimo de `estado de salida`.

`señal`
: Un mecanismo por el cual un proceso puede ser notificado por el núcleo de un evento ocurrido en el sistema.

`instrucción integrada especial`
: Una instrucción integrada del intérprete que ha sido clasificada como especial por el estándar POSIX.

`símbolo`
: Una secuencia de caracteres considerada como unidad única por el intérprete. O bien es una `palabra` o un `operador`

`palabra`
: Una secuencia de caracteres tratada como una unidad por el intérprete. Las palabras no pueden incluir `metacaracteres` sin entrecomillar.

3 Funcionalidades Básicas del Intérprete

Bash es un acrónimo para '`Bourne-Again SHell`'. El Bourne shell es el intérprete tradicional de Unix escrito originalmente por Stephen Bourne. Todas las instrucciones integradas del Bourne shell están disponibles en Bash. Las reglas para evaluación y entrecomillado se toman de la especificación POSIX para el intérprete Unix 'estándar'.

Este capítulo resume brevemente los 'bloques constructores' del intérprete: instrucciones, estructuras de control, funciones del intérprete, *parámetros* del intérprete, expansiones del intérprete, *redirecciones*, las cuales son una forma de dirigir entrada y salida desde y hacia archivos nombrados, y cómo el intérprete ejecuta instrucciones.

3.1 Sintaxis del Intérprete

Cuando el intérprete lee la entrada, procede a través de una secuencia de operaciones. Si la entrada indica el inicio de un comentario, el intérprete ignora el símbolo de comentario ('`#`') y el resto de esa línea.

De lo contrario, básicamente, el intérprete lee su entrada y divide la entrada en palabras y operadores, empleando las reglas de entrecomillado para elegir qué significados asignar a varias palabras y caracteres.

El intérprete entonces transforma estos símbolos en instrucciones y otras construcciones, elimina el significado especial de ciertas palabras o caracteres, expande otros, redirige entrada y salida según sea necesario, ejecuta la instrucción especificada, espera el estado de salida de la instrucción y deja disponible ese estado de salida para inspección o procesamiento posterior.

3.1.1 Funcionamiento del Intérprete

Lo siguiente es una breve descripción del funcionamiento del intérprete cuando lee y ejecuta una instrucción. Básicamente, el intérprete hace lo siguiente:

1. Lee su entrada de un archivo (véase Sección 3.8 [Guiones del Intérprete], página 46), de una cadena proporcionada como un argumento para la opción de llamada `-c` (véase Sección 6.1 [Llamando a Bash], página 95) o de la terminal del usuario.
2. Divide la entrada en palabras y operadores, atendiendo a las reglas de entrecomillado descritas en Sección 3.1.2 [Entrecomillado], página 6. Estos símbolos son separados por `metacaracteres`. La expansión de alias se realiza por este paso (véase Sección 6.6 [Aliases], página 105).
3. Transforma estos símbolos en instrucciones simples y compuestas (véase Sección 3.2 [Instrucciones del Intérprete], página 8).
4. Realiza las distintas expansiones del intérprete (véase Sección 3.5 [Expansiones del Intérprete], página 24), dividiendo los símbolos expandidos en listas de nombres de archivo (véase Sección 3.5.8 [Expansión de Nombre de Archivo], página 35, e instrucciones y argumentos.
5. Realiza cualquier redirección necesaria (véase Sección 3.6 [Redirecciones], página 37) y elimina los operadores de redirección y sus operandos de la lista de argumentos.
6. Ejecuta la instrucción (véase Sección 3.7 [Ejecutando Instrucciones], página 42).
7. Opcionalmente espera a que la instrucción termine y recoge su estado de salida (véase Sección 3.7.5 [Estado de Salida], página 45).

3.1.2 Entrecomillado

El entrecomillado se usa para eliminar el significado especial para el intérprete de ciertos caracteres o palabras. El entrecomillado se puede usar para deshabilitar el tratamiento especial de caracteres especiales, para evitar que se reconozcan palabras reservadas como tales y para evitar la expansión de parámetro.

Cada uno de los metacaracteres del intérprete (véase Capítulo 2 [Definiciones], página 3) tiene un significado especial para el intérprete y debe ser entrecomillado si se representa a sí mismo. Cuando se usan las facilidades de expansión del historial de instrucciones (véase Sección 9.3 [Interacción con el Historial], página 163), el carácter de *expansión de historial*, normalmente '`!`', debe entrecomillarse para evitar la expansión de historial. Véase Sección 9.1 [Servicios del Historial de Bash], página 161, para más detalles sobre la expansión de historial.

Hay tres mecanismos de entrecomillado: el *carácter de escape*, comillas simples y comillas dobles.

3.1.2.1 Carácter de Escape

La barra invertida sin entrecomillar '`\`' es el carácter de escape de Bash. Preserva el valor literal del siguiente carácter que lo sigue, con la excepción de `nueva línea`. Si aparece una pareja `\nueva línea` y la barra invertida en sí no está entrecomillada, la `\nueva línea` se trata como una continuación de línea (es decir, es eliminada del flujo de entrada y es efectivamente ignorada).

3.1.2.2 Comillas Simples

Encerrar caracteres en comillas simples ('`'`') preserva el valor literal de cada carácter dentro de las comillas. Una comilla simple no puede encontrarse entre comillas simples, incluso precedida de una barra invertida.

3.1.2.3 Comillas Dobles

Encerrar caracteres en comillas dobles ('`"`') preserva el valor literal de todos los caracteres entre las comillas, con la excepción de '`$`', '`` ` ``', '`\`' y, cuando está activada la expansión de historial, '`!`'. Cuando el intérprete está en modo POSIX (véase Sección 6.11 [Modo POSIX de Bash], página 111), el '`!`' no tiene un significado especial entre comillas dobles, incluso cuando está activada la expansión de historial. Los caracteres '`$`' y '`` ` ``' conservan su significado especial dentro de comillas dobles (véase Sección 3.5 [Expansiones del Intérprete], página 24). La barra invertida solo conserva su significado especial seguido por uno de los siguientes caracteres: '`$`', '`` ` ``', '`"`', '`\`' o `nueva línea`. Dentro de comillas dobles, las barras invertidas que son seguidas por uno de estos caracteres son eliminadas. Las barras invertidas precediendo caracteres sin un significado especial no se modifican. Una comilla doble puede ser entrecomillada dentro de comillas dobles precediéndola con una barra invertida. Si está habilitada, la expansión del historial se realizará a no ser que se escape un '`!`' que aparezca entre comillas dobles con una barra invertida. La barra invertida que precede al '`!`' no es eliminada.

Los parámetros especiales '`*`' y '`@`' tienen significado especial entre comillas dobles (véase Sección 3.5.3 [Expansión de Parámetros del Intérprete], página 27).

3.1.2.4 Entrecomillado ANSI-C

Las palabras de forma `$'cadena'` se tratan de forma especial. La palabra se expande a *cadena*, con los caracteres escapados por barras invertidas reemplazados como se especifica en el estándar ANSI C. Si están presentes, las secuencias de escape de barras invertidas se decodifican así:

`\a` alerta (timbre)

`\b` retroceso

`\e`
`\E` un carácter de escape (no ANSI C)

`\f` salto de página

`\n` nueva línea

`\r` retorno de carro

`\t` tabulación horizontal

`\v` tabulación vertical

`\\` barra invertida

`\'` comilla simple

`\"` comilla doble

`\?` símbolo de interrogación de cierre

`\`*nnn* el carácter de ocho bits cuyo valor es el valor octal *nnn* (de uno a tres dígitos octales).

`\x`*HH* el carácter de ocho bits cuyo valor es el valor hexadecimal *HH* (uno o dos dígitos hexadecimales)

`\u`*HHHH* el carácter Unicode (ISO/IEC 10646) cuyo valor es el valor hexadecimal *HHHH* (de uno a cuatro dígitos hexadecimales)

`\U`*HHHHHHHH*
el carácter Unicode (ISO/IEC 10646) cuyo valor es el valor hexadecimal *HHHHHHHH* (de uno a ocho dígitos hexadecimales)

`\c`*x* un catácter control-*x*

El resultado expandido es entrecomillado con una comilla, como si el símbolo de dolar no hubiera estado presente.

3.1.2.5 Traducción de Región Específica

Una cadena entre comillas dobles precedida del signo de dolar (‘`$`’) hará que la cadena sea traducida según la configuración regional actual. La infraestructura de *gettext* realiza la búsqueda y la traducción del catálogo de mensajes, usando las variables del intérprete `LC_MESSAGES` y `TEXTDOMAIN`, como se explica a continuación. Si la configuración regional actual es `C` o `POSIX`, se ignora el signo de dolar. Si la cadena se traduce y reemplaza, el remplazo está entre comillas dobles.

Algunos sistemas usan el catálogo de mensajes elegido por la variable del intérprete `LC_MESSAGES`. Otros crean el nombre del catálogo de mensajes desde el valor de la variable del intérprete `TEXTDOMAIN`, posiblemente añadiendo un sufijo de '`.mo`'. Si usa la variable `TEXTDOMAIN`, puede que necesite establecer la variable `TEXTDOMAINDIR` a la ubicación del archivo de catálogo de mensajes. Otros usan todavía ambas variables de esta forma: `TEXTDOMAINDIR`/`LC_MESSAGES`/LC_MESSAGES/`TEXTDOMAIN`.mo.

3.1.3 Comentarios

En un intérprete no interactivo o un intérprete interactivo en que la opción `interactive_comments` para la instrucción integrada `shopt` está activada (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73), una palabra que empieza por '`#`' hace que esa palabra y todos los caracteres restantes en esa línea sean ignorados. Un intérprete interactivo sin la opción `interactive_comments` activada no permite comentarios. La opción `interactive_comments` está activada por defecto en intérpretes interactivos. Véase Sección 6.3 [Intérpretes Interactivos], página 99, para una descripción de qué hace a un intérprete interactivo.

3.2 Instrucciones del Intérprete

Una instrucción simple del intérprete como `echo a b c` consta de la instrucción en sí seguida por argumentos, separados por espacios.

Las instrucciones del intérprete más complejas están compuestas de instrucciones simples organizadas conjuntamente de diferentes maneras: en una tubería en que la salida de una instrucción se convierte en la entrada de una segunda, en un bucle o instrucción condicional o en alguna otra agrupación.

3.2.1 Palabras Reservadas

Palabras reservadas que tienen un significado especial para el intérprete. Se usan para empezar y terminar las instrucciones compuestas del intérprete.

Las siguientes palabras son reconocidas como reservadas cuando no están entrecomilladas y la primera palabra de una instrucción (vea más abajo las excepciones):

```
if        then    elif    else    fi        time
for       in      until   while   do        done
case      esac    coproc  select  function
{         }       [[      ]]      !
```

`in` se reconoce como una palabra reservada si es la tercera palabra de una instrucción `case` o `select`. `in` y `do` se reconocen como palabras reservadas si son la tercera palabra de una instrucción `for`.

3.2.2 Instrucciones Simples

Una instrucción simple es el tipo de instrucción encontrado más a menudo. Es solo una secuencia de palabras separadas por `blanco`s, terminada en uno de los operadores de control del intérprete (véase Capítulo 2 [Definiciones], página 3). La primera palabra especifica generalmente una instrucción para que sea ejecutada, siendo el resto de las palabras los argumentos de esa instrucción.

El estado de retorno (véase Sección 3.7.5 [Estado de Salida], página 45) de una instrucción simple es su estado de salida tal como estipula la función POSIX 1003.1 `waitpid` o 128+*n* si la instrucción fue terminada por la señal *n*.

3.2.3 Tuberías

Una `tubería` es una secuencia de una o más instrucciones separadas por uno de los operadores de control '`|`' o '`|&`'.

El formato para una tubería es

```
[time [-p]] [!] instrucción1 [ | o |& instrucción2 ] ...
```

La salida de cada instrucción en la tubería es conectada por medio de una tubería a la entrada de la siguiente instrucción. Es decir, cada instrucción lee la salida de la anterior instrucción. Esta conexión se realiza antes de cualquier redirección especificada por la instrucción.

Si se usa '`|&`', el error estándar de *instrucción1*, además de su salida estándar, es conectado a la entrada estándar de *instrucción2* a través de la tubería; es una abreviatura de `2>&1 |`. Esta redirección implícita del error estándar a la salida estándar es realizada después de cualquier redirección especificada por la instrucción.

La palabra reservada `time` hace que sean imprimidas estadísticas temporales para la tubería una vez finalice. Las estadísticas constan actualmente del tiempo (de reloj) transcurrido y del tiempo de usuario y de sistema consumido por la ejecución de la instrucción. La opción `-p` cambia el formato de salida al especificado por POSIX. Cuando el intérprete está en el modo POSIX (véase Sección 6.11 [Modo POSIX de Bash], página 111), no reconoce `time` como una palabra reservada si el siguiente símbolo comienza por '`-`'. La variable `TIMEFORMAT` puede ser establecida a una cadena de formato que especifica cómo debe ser mostrada la información temporal. Véase Sección 5.2 [Variables de Bash], página 82, para una descripción de los formatos disponibles. El uso de `time` como una palabra reservada permite el cronometraje de las instrucciones integradas del intérprete, funciones del intérprete y tuberías. Una instrucción `time` externa no puede cronometrar estas fácilmente.

Cuando el intérprete está en modo POSIX (véase Sección 6.11 [Modo POSIX de Bash], página 111), `time` puede ser seguido por una nueva línea. En este caso, el intérprete muestra el tiempo total de usuario y de sistema consumido por el intérprete y sus hijos. La variable `TIMEFORMAT` puede ser usada para especificar el formato de la información temporal.

Si la tubería no es ejecutada asíncronamete (véase Sección 3.2.4 [Listas], página 10), el intérprete espera a que todas las instrucciones en la tubería se completen.

Cada instrucción en una tubería se ejecuta en su propio subintérprete, que es un proceso separado (véase Sección 3.7.3 [Entorno de Ejecución de Instrucciones], página 43). Si la opción `lastpipe` está activada por medio de la instrucción integrada `shopt` (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73), el último elemento de una tubería puede ser ejecutado por el proceso del intérprete.

El estado de salida de una tubería es el estado de salida de la última instrucción en la tubería, a no ser que esté activada la opción `pipefail` (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68). Si `pipefail` está habilitada, el estado de retorno de la tubería es el valor de la última (la más a la derecha) instrucción que finalice con un estado distinto a cero, o cero si todas las instrucciones finalizan con éxito. Si la palabra reservada '`!`' precede a la tubería, el estado de salida es la negación lógica del estado de salida descrito

anteriormente. El intérprete espera a que finalicen todas las instrucciones en la tubería antes de devolver un valor.

3.2.4 Listas de Instrucciones

Una `lista` es una secuencia de una o más tuberías separadas por uno de los operadores '`;`', '`&`', '`&&`' o '`||`', y opcionalmente terminada por '`;`', '`&`', or a `nueva línea`.

De esta lista de operadores, '`&&`' y '`||`' tienen igual precedencia, seguidos de '`;`' y '`&`', que tienen igual precedencia.

Una secuencia de una o más nuevas líneas puede aparecer en una `lista` para delimitar instrucciones, equivalente a un punto y coma.

Si se termina una instrucción con el operador de control '`&`', el intérprete ejecuta la instrucción asíncronamente en un subintérprete. Esto se conoce como ejecutar la instrucción en *segundo plano*, y estas se conocen como instrucciones *asíncronas*. El intérprete no espera a que termine la instrucción, y el estado de retorno es 0 (verdadero). Cuando el control de tareas no está activo (véase Capítulo 7 [Control de Tareas], página 118), la entrada estándar para instrucciones asíncronas, en ausencia de cualquier redirección explícita, es redirigida desde `/dev/null`.

Las instrucciones separadas por un '`;`' son ejecutadas secuencialmente; el intérprete espera a que cada instrucción termine en orden. El estado de retorno es el estado de salida de la última instrucción ejecutada.

Las listas AND y OR son secuencias de una o más tuberías separadas por los operadores de control '`&&`' y '`||`', respectivamente. Las listas AND y OR son ejecutadas con asociatividad izquierda.

Una lista AND tiene la forma

```
instrucción1 && instrucción2
```

instrucción2 es ejecutada únicamente si *instrucción1* devuelve un estado de salida de cero (éxito).

Una lista OR tiene la forma

```
instrucción1 || instrucción2
```

instrucción2 es ejecutada únicamente si *instrucción1* devuelve un estado de salida distinto de cero.

El estado de retorno de las listas AND y OR es el estado de salida de la última instrucción ejecutada en la lista.

3.2.5 Instrucciones Compuestas

Las instrucciones compuestas son las construcciones del lenguaje de programación del intérprete. Cada construcción empieza con una palabra reservada u operador de control y termina con una palabra reservada u operador correspondiente. Cualquier redirección (véase Sección 3.6 [Redirecciones], página 37) asociada a una instrucción compuesta se aplica a todas las instrucciones dentro de esa instrucción compuesta a no ser que sea reemplazada explícitamente.

En la mayoría de los casos, una lista de instrucciones en una descripción de instrucción compuesta puede ser separada del resto de las instrucción por una o más nuevas líneas, y puede ser seguida por una nueva línea en lugar de un punto y coma.

Bash proporciona construcciones de bucle, instrucciones condicionales y mecanismos para agrupar instrucciones y ejecutarlas como una unidad.

3.2.5.1 Construcciones de Bucle

Bash acepta las siguientes construcciones de bucle.

Tenga en cuenta que cada vez que aparezca un '`;`' en la descripción de la sintaxis de una instrucción, puede ser reemplazado por una o más nuevas líneas.

`until` La sintaxis de la instrucción `until` es:

```
until instrucciones-test; do instrucciones-consiguientes; done
```

Ejecuta *instrucciones-consiguientes* mientras que *instrucciones-test* tenga un estado de salida que no sea cero. El estado de retorno es el estado de salida de la última instrucción ejecutada en *instrucciones-consiguientes*, o cero si ninguna fue ejecutada.

`while` La sintaxis de la instrucción `while` es:

```
while instrucciones-test; do instrucciones-consiguientes; done
```

Ejecuta *instrucciones-consiguientes* mientras que *instrucciones-test* tenga un estado de salida de cero. El estado de retorno es el estado de salida de la última instrucción ejecutada en *instrucciones-consiguientes*, o cero si ninguna fue ejecutada.

`for` La sintaxis de la instrucción `for` es:

```
for nombre [ [in [palabras ...] ] ; ] do instrucciones; done
```

Expande *palabras* (véase Sección 3.5 [Expansiones del Intérprete], página 24) y ejecuta *instrucciones* una vez por cada elemento en la lista resultante, con *nombre* vinculado al miembro actual. Si '`in palabras`' no está presente, la instrucción `for` ejecuta las *instrucciones* una vez por cada parámetro posicional que esté establecido, como si '`in "$@"`' hubiera sido especificado (véase Sección 3.4.2 [Parámetros Especiales], página 23).

El estado de retorno es el estado de salida de la última instrucción que ejecuta. Si no hay elementos en la expansión de *palabras*, no se ejecutan instrucciones y el estado de retorno es cero.

También se acepta una forma alternativa de la instrucción `for`:

```
for (( expr1 ; expr2 ; expr3 )) ; do instrucciones ; done
```

Primero, la expresión aritmética *expr1* se evalúa de acuerdo a las reglas descritas adelante (véase Sección 6.5 [Aritmética del Intérprete], página 103). La expresión aritmética *expr2* es entonces evaluada repetidamente hasta que evalúe a cero. Cada vez que *expr2* evalúe a un valor distinto de cero, se ejecutan las *instrucciones* y se evalúa la expresión aritmética *expr3*. Si se omite cualquier expresión, se comporta como si evaluara a 1. El valor de retorno es el estado de salida de la última instrucción que es ejecutada en *instrucciones*, o falso si cualquiera de las expresiones es inválida.

Las instrucciones integradas `break` y `continue` (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48) se pueden usar para controlar la ejecución del bucle.

3.2.5.2 Construcciones Condicionales

`if` La sintaxis de la instrucción `if` es:

```
if instrucciones-test; then
  instrucciones-consiguientes;
[elif más-instrucciones-test; then
  más-consiguientes;]
[else consiguientes-alternativas;]
fi
```

La lista de *instrucciones-test* es ejecutada, y si su estado de retorno es cero, la lista de *instrucciones-consiguientes* es ejecutada. Si *instrucciones-test* devuelve un estado de retorno distinto de cero, cada lista `elif` es ejecutada en orden, y si su estado es cero, el *más-consiguientes* correspondiente es ejecutado y la instrucción se completa. Si '`else` *consiguientes-alternativas*' está presente, y la instrucción final en la última cláusula `if` o `elif` tiene un estado de salida distinto de cero, entonces *consiguientes-alternativas* es ejecutado. El estado de retorno es el estado de salida de la última instrucción ejecutada, o cero si ninguna condición se evaluó a verdadero.

`case` La sintaxis de la instrucción `case` es:

```
case palabra in
    [ [(] patrón [| patrón]...) lista-de-instrucciones ;;]...
esac
```

`case` ejecutará selectivamente la *lista-de-instrucciones* correspondiente al primer *patrón* que coincide con *palabra*. La coincidencia se realiza según las reglas descritas más adelante en Sección 3.5.8.1 [Coincidencia de Patrones], página 36. Si la opción del intérprete `nocasematch` (vea la descripción de `shopt` en Sección 4.3.2 [La Instrucción Integrada Shopt], página 73) está habilitada, la coincidencia se realiza sin tener en cuenta las mayúsculas y minúsculas de caracteres alfabéticos. El '`|`' es usado para separar múltiples patrones, y el operador '`)`' finaliza una lista de patrón. Se llama *cláusula* a la lista de patrones y a su lista de instrucciones asociada.

Cada cláusula debe terminar en '`;;`', '`;&`' o '`;;&`'. La *palabra* pasa por la expansión de virgulilla, expansión de parámetro, sustitución de instrucción, expansión aritmética y eliminación de comillas (véase Sección 3.5.3 [Expansión de Parámetros del Intérprete], página 27) antes de que se intente la coincidencia. Cada *patrón* pasa por la expansión de virgulilla, sustitución de instrucción y expansión aritmética.

Puede haber un número arbitrario de cláusulas `case`, cada una terminada en '`;;`', '`;&`' o '`;;&`'. El primer patrón que coincide determina la lista de instrucciones que es ejecutada. Es una expresión normal usar '`*`' como el patrón final para definir el caso predeterminado, ya que ese patrón siempre coincidirá.

Aquí hay un ejemplo que usa `case` en un guion que podría ser usado para describir un rasgo interesante de un animal:

```
echo -n "Introduzca el nombre de un animal: "
read ANIMAL
```

```
echo -n "El $ANIMAL tiene "
case $ANIMAL in
  caballo | perro | gato) echo -n "cuatro";;
  hombre | canguro ) echo -n "dos";;
  *) echo -n "un número desconocido de";;
esac
echo " patas."
```

Si se usa el operador '`;;`', no se intentan coincidencias posteriores después de la primera coincidencia de patrón. Usar '`;&`' en lugar de '`;;`' hace que la ejecución continúe con la *lista-de-instrucciones* asociada a la siguiente cláusula, si hay. Usar '`;;&`' en lugar de '`;;`' hace que el intérprete compruebe los patrones en la siguiente cláusula, si hay, y ejecute cualquier *lista-de-instrucciones* asociada cuando haya una coincidencia exitosa, continuando la ejecución de la declaración `case` como si el patrón no hubiera coincidido.

El estado de retorno es cero si ningún *patrón* coincide. En caso contrario, el estado de retorno es el estado de salida de la *lista-de-instrucciones* ejecutada.

`select`

La construcción `select` permite la sencilla generación de menús. Tiene casi la misma sintaxis que la instrucción `for`:

```
select nombre [in palabras ...]; do instrucciones; done
```

La lista de palabras detrás de `in` es expandida, generando una lista de elementos. El conjunto de palabras expandidas es imprimido en el flujo de la salida de error estándar, cada una precedida por un número. Si se omite el '`in palabras`', se imprimen los parámetros posicionales, como si '`in "$@"`' hubiera sido especificado. El prompt `PS3` es entonces mostrado y es leída una línea de la entrada estándar. Si la línea consta de un número correspondiente a una de las palabras mostradas, el valor de *nombre* es establecido a esa palabra. Si la línea está vacía, las palabras y el prompt se muestran de nuevo. Si es leído `EOF`, la instrucción `select` se completa. Cualquier otro valor leído hace que *nombre* se establezca a nulo. La línea leída es guardada en la variable `REPLY`.

Las *instrucciones* se ejecutan después de cada selección hasta que se ejecute una instrucción `break`, en cuyo momento la instrucción `select` se completa.

He aquí un ejemplo que permite al usuario escoger un nombre de archivo del directorio actual y muestra el nombre y el índice del archivo seleccionado.

```
select narchivo in *;
do
echo escogiste $narchivo \($REPLY\)
break;
done
```

`((...))`

```
(( expresión ))
```

La *expresión* aritmética se evalúa de acuerdo a las reglas descritas adelante (véase Sección 6.5 [Aritmética del Intérprete], página 103). Si el valor de la

expresión es distinto de cero, el estado de retorno es 0; de lo contrario, el valor de retorno es 1. Esto es exactamente equivalente a

```
let "expresión"
```

Véase Sección 4.2 [Instrucciones Integradas de Bash], página 56, para una descripción completa de la instrucción integrada `let`.

`[[...]]`

```
[[ expresión ]]
```

Devuelve un estado de 0 o 1 dependiendo de la evaluación de la expresión condicional *expresión*. Las expresiones están compuestas de los elementos primarios descritos más adelante en Sección 6.4 [Expresiones Condicionales de Bash], página 101. La división de palabras y la expansión de nombre de archivo no se realizan en las palabras entre el `[[` y el `]]`; son realizadas la expansión de virgulilla, la expansión de parámetro y de variable, la expansión aritmética, la sustitución de instrucciones, la sustitución de proceso y la eliminación de entrecomillado. Los operadores condicionales como '`-f`' no deben estar entrecomillados para ser reconocidos como opciones primarias.

Cuando se usa con `[[`, los operadores '`<`' y '`>`' ordenan lexicográficamente usando la configuración regional actual.

Cuando se usan los operadores '`==`' y '`!=`', la cadena a la derecha del operador es considerada un patrón y es correspondida según las reglas descritas a continuación en Sección 3.5.8.1 [Coincidencia de Patrones], página 36, como si la opción del intérprete `extglob` estuviera habilitada. El operador '`=`' es idéntico a '`==`'. Si la opción del intérprete `nocasematch` (vea la descripción de `shopt` en Sección 4.3.2 [La Instrucción Integrada Shopt], página 73) está habilitada, la correspondencia se realiza sin importar las mayúsculas y minúsculas de los caracteres alfabéticos. El valor de retorno es 0 si la cadena coincide ('`==`') o no coincide ('`!=`') con el patrón, y 1 en caso contrario. Cualquier parte del patrón puede ser entrecomillada para forzar la porción entrecomillada a que sea coincidida con una cadena.

Está disponible un operador binario, '`=~`', con la misma precedencia que '`==`' y '`!=`'. Cuando se usa, la cadena a la derecha del operador es considerada como una expresión regular extendida POSIX y se comprueba el patrón según esto (usando las interfaces POSIX `regcomp` y `regexec` descritas en *regex*(3)). El valor de retorno es 0 si la cadena coincide con el patrón, 1 en caso contrario. Si la expresión regular es sintácticamente incorrecta, el valor de retorno de la expresión condicional es 2. Si la opción del intérprete `nocasematch` (consulte la descripción de `shopt` en ⟨undefined⟩ [The Shopt Builtin], página ⟨undefined⟩) está habilitada, la coincidencia se realiza sin tener en cuenta si los caracteres alfabéticos están en minúscula o en mayúscula. Se puede entrecomillar cualquier parte del patrón para forzar que la parte entrecomillada sea coincidida como una cadena. Las expresiones de llaves en expresiones regulares deben ser tratadas con cuidado, puesto que los caracteres de entrecomillado normales pierden sus significados entre llaves. Si el patrón se guarda en una variable del intérprete, entrecomillar la expansión de variable fuerza la coincidencia del patrón entero como una cadena.

El patrón coincidirá si coincide con cualquier parte de la cadena. Anclar el patrón usando los operadores de expresión regular '`^`' y '`$`' lo fuerzan a que coincida con la cadena entera. La variable de vector `BASH_REMATCH` graba qué partes de la cadena coincidieron con el patrón. El elemento de `BASH_REMATCH` con el índice 0 contiene la porción de la cadena que coincide con la expresión regular entera. Las subcadenas coincididas por subexpresiones entre paréntesis dentro de la expresión regular son guardadas en los índices restantes de `BASH_REMATCH`. El elemento de `BASH_REMATCH` de índice *n* es la porción de la cadena que coincide con la subexpresión número *n* entre paréntesis.

Por ejemplo, lo siguiente coincidirá con una línea (guardada en la variable del intérprete *línea*) si hay una secuencia de caracteres en cualquier lugar del valor que conste de cualquier número, incluido cero, de caracteres de `espacio`, cero o más instancias de '`a`', después una '`b`':

```
[[ $linea =~ [[:space:]]*(a)?b ]]
```

Eso quiere decir que valores como '`aab`' y ' `aaaaaab`' coincidirán, como también lo hará una línea que contenga una '`b`' en cualquier lugar en su valor.

Guardar la expresión regular en una variable del intérprete es habitualmente una forma útil de evitar problemas con entrecomillar caracteres que son especiales para el intérprete. A veces es difícil especificar una expresión regular literalmente sin usar comillas, o llevar un seguimiento del entrecomillado usado por las expresiones regulares mientras se presta atención a la eliminación de comillas del intérprete. Usar una variable del intérprete disminuye estos problemas. Por ejemplo, lo siguiente es equivalente a lo anterior:

```
patron='[[:space:]]*(a)?b'
[[ $linea =~ $patron ]]
```

Si quieres hacer coincidir un carácter que es especial para la gramática de expresiones regulares, tiene que ser entrecomillado para eliminar su significado especial. Esto significa que en el patrón '`xxx.txt`', el '`.`' corresponde a cualquier carácter en la cadena (su significado usual de expresión regular), pero en el patrón '`"xxx.txt"`' puede corresponder solo a un '`.`' literal. Los programadores del intérprete deberían tener un especial cuidado con las barras invertidas, puesto que las barras invertidas son usadas tanto por el intérprete como por las expresiones regulares para eliminar el significado especial del siguiente carácter. Los siguientes dos conjuntos de instrucciones *no* son equivalentes:

```
patron='\.'

[[ . =~ $patron ]]
[[ . =~ \. ]]

[[ . =~ "$patron" ]]
[[ . =~ '\.' ]]
```

Las primeras dos coincidencias tendrán éxito, pero las segundas dos no, porque en las segundas las barras invertidas serán parte del patrón que será coincidido. En los primeros dos ejemplos, la barra invertida elimina el significado especial de '`.`', para que el '`.`' coincida. Si la cadena en los primeros ejemplos fuera

cualquier cosa distinta de '`.`', digamos que '`a`', el patrón no coincidiría, porque el '`.`' entrecomillado en el patrón pierde su significado especial de coincidir con cualquier carácter.

Las expresiones pueden ser combinadas usando los siguientes operadores, listados en orden decreciente de precedencia:

`( `*expresión*` )`
: Devuelve el valor de *expresión*. Esto puede ser usado para sobrescribir la precedencia normal de operadores.

`! `*expresión*
: Verdadero si *expresión* es falso.

expresión1 `&&` *expresión2*
: Verdadero si tanto *expresión1* como *expresión* son verdaderas.

expresión1 `||` *expresión2*
: Verdadero si *expresión1* o *expresión2* es verdadera.

Los operadores `&&` y `||` no evalúan *expresión2* si el valor de *expresión1* es suficiente para determinar el valor de retorno de la expresión condicional completa.

3.2.5.3 Agrupación de Instrucciones

Bash proporciona dos maneras de agrupar una lista de instrucciones para ser ejecutada como una únidad. Cuando las instrucciones son agrupadas, las redirecciones pueden ser agrupadas por la lista de instrucciones completa. Por ejemplo, la salida de todas las instrucciones en la lista puede ser redirigida a un único flujo.

`()`

`( `*lista*` )`

Ubicar una lista de instrucciones entre paréntesis hace que se cree un entorno de subintérprete (véase Sección 3.7.3 [Entorno de Ejecución de Instrucciones], página 43), y cada una de las instrucciones en *lista* sea ejecutada en ese subintérprete. Dado que la *lista* es ejecutada en un subintérprete, las asignaciones de variable no permanecen en vigor después de que el subintérprete finalice.

`{}`

`{ `*lista*`; }`

Ubicar una lista de instrucciones entre llaves hace que la lista sea ejecutada en el contexto actual del intérprete. No se crea un subintérprete. Se requiere el punto y coma (o nueva línea) después de *lista*.

Además de la creación de un subintérprete, hay una sutil diferencia entre estas dos construcciones debido a razones históricas. Las llaves son `palabras reservadas`, así que tienen que estar separadas de *lista* por `blanco`s u otros metacaracteres del intérprete. Los paréntesis son `operadores`, y son reconocidos como símbolos separados por el intérprete incluso si no están separados de la *lista* por espacios en blanco.

El estado de salida de estas dos construcciones es el estado de salida de *lista*.

3.2.6 Coprocesos

Un `coproceso` es una instrucción del intérprete precedida por la palabra reservada `coproc`. Un coproceso es ejecutado asíncronamente en un subintérprete, como si la instrucción se hubiera terminado con el operador de control '`&`', con una tubería de dos sentidos establecida entre el intérprete en ejecución y el coproceso.

El formato para un coproceso es:

```
coproc [NOMBRE] instrucción [redirecciones]
```

Esto crea un coproceso llamado *NOMBRE*. Si no se proporciona *NOMBRE*, el nombre predeterminado es *COPROC*. *NOMBRE* no tiene que ser proporcionado si *instrucción* es una instrucción simple (véase Sección 3.2.2 [Instrucciones Simples], página 8); de lo contrario, se interpreta como la primera palabra de una instrucción simple.

Cuando se ejecuta el coproceso, el intérprete crea una variable de vector (véase Sección 6.7 [Vectores], página 105) llamada `NAME` en el contexto del intérprete en ejecución. La salida estándar de *instrucción* se conecta a través de una tubería al descriptor de archivo en el intérprete en ejecución, y ese descriptor de archivo es asignado a `NAME`[0]. La entrada estándar de *instrucción* se conecta a través de una tubería a un descriptor de archivo en el intérprete en ejecución, y ese descriptor de archivo es asignado a `NAME`[1]. Esta tubería se establece antes de cualquier redirección especificada por la instrucción (véase Sección 3.6 [Redirecciones], página 37). Los descriptores de archivo pueden ser utilizados como argumentos para instrucciones del intérprete y redirecciones usando expansiones de palabras estándares. Aparte de aquellos creados para ejecutar sustituciones de instrucciones y procesos, los descriptores de archivos no están disponibles en subintérpretes.

El identificador de proceso del intérprete generado para ejecutar el coproceso está disponible como el valor de la variable *NAME*_PID. La instrucción integrada `wait` puede usarse para esperar a que el coproceso finalice.

Dado que el coproceso es creado como una instrucción asíncrona, la instrucción `coproc` siempre retorna éxito. El estado de retorno de un coproceso es el estado de salida de *instrucción*.

3.2.7 GNU Parallel

Hay varias maneras de ejecutar instrucciones en paralelo que no están integradas en Bash. GNU Parallel es una herramienta para hacer precisamente esto.

GNU Parallel, como su nombre sugiere, puede ser usado para construir y ejecutar instrucciones en paralelo. Puede ejecutar las mismas instrucciones con diferentes argumentos, ya sean nombres de archivo, nombre de usuario, nombres de equipos o líneas leídas de archivos. GNU Parallel proporciona referencias abreviadas a muchas de las operaciones más comunes (líneas de entrada, varias porciones de la línea de entrada, diferentes formas de especificar la fuente de entrada y demás). Parallel puede reemplazar `xargs` o surtir de instrucciones de sus fuentes de entrada a varias instancias diferentes de Bash.

Para una descripción completa, consulte la documentación de GNU Parallel. Unos pocos ejemplos deberían proporcionar una breve introducción a su uso.

Por ejemplo, es fácil reemplazar `xargs` para comprimir con gzip todos los archivos HTML en el directorio actual y sus subdirectorios:

```
find . -type f -name '*.html' -print | parallel gzip
```

Si necesita proteger caracteres especiales como nuevas líneas en nombres de archivo, use la opción de find `-print0` y la opción de parallel `-0`.

Puede usar Parallel para mover archivos desde el directorio actual cuando el número de archivos es muy grande para procesar con una llamada a `mv`:

```
printf '%s\n' * | parallel mv {} dirdest
```

Como puede ver, el {} es reemplazado por cada línea leída de la entrada estándar. Aunque usar `ls` funcionará en la mayoría de casos, no es bastante para manejar todos los nombres de archivo. `printf` es una instrucción integrada del intérprete, y por ello no está sujeta a los límites del núcleo en cuanto al número de argumentos para un programa, de forma que puede usar '`*`' (pero lea a continuación sobre la opción del intérprete `dotglob`). Si necesita acomodar caracteres especiales en nombres de archivo, puede usar

```
printf '%s\0' * | parallel -0 mv {} dirdest
```

como se menciona anteriormente.

Esto ejecutará tantas instrucciones `mv` como archivos haya en el directorio actual. Puede emular un `xargs` paralelo añadiendo la opción `-X`:

```
printf '%s\0' * | parallel -0 -X mv {} dirdest
```

(Puede que tenga que modificar el patrón si tiene activada la opción `dotglob`.)

GNU Parallel puede reemplazar ciertas expresiones comunes que operan en líneas leídas de un archivo (en este caso, los nombres de archivo listados uno por línea):

```
while IFS= read -r x; do
haz-algo1 "$x" "config-$x"
haz-algo2 < "$x"
done < archivo | procesa-salida
```

con una sintaxis más compleja reminiscente de lambdas:

```
cat lista | parallel "haz-algo1 {} config-{} ; haz-algo2 < {}" |
           procesa-salida
```

Parallel proporciona un mecanismo integrado para eliminar las extensiones de nombre de archivo, que se presta a transformaciones o renombraciones de archivos en lote:

```
ls *.gz | parallel -j+0 "zcat {} | bzip2 >{.}.bz2 && rm {}"
```

Esto volverá a comprimir todos los archivos en el directorio actual con nombres acabados en .gz usando bzip2, ejecutando una tarea por CPU (-j+0) en paralelo. (Usamos `ls` por brevedad aquí; usar `find` como antes es más robusto de cara a nombres de archivo que contienen caracteres inesperados.) Parallel puede tomar argumentos de la línea de instrucciones; lo anterior puede ser también escrito como

```
parallel "zcat {} | bzip2 >{.}.bz2 && rm {}" ::: *.gz
```

Si una instrucción genera salida, puede que quieras preservar el orden de entrada en la salida. Por ejemplo, la siguiente instrucción

```
{
    echo foss.org.my ;
    echo debian.org ;
    echo freenetproject.org ;
} | parallel traceroute
```

mostrará como salida la llamada a traceroute que finalice primero. Añadiendo la opción `-k`

```
{
    echo foss.org.my ;
    echo debian.org ;
    echo freenetproject.org ;
} | parallel -k traceroute
```

asegurará que la salida de `traceroute foss.org.my` sea mostrada primero.

Finalmente, Parallel se puede usar para ejecutar una secuencia de instrucciones del intérprete en paralelo, similar a '`cat file | bash`'. No es raro tomar una lista de nombres de archivo, crear conjuntos de instrucciones del intérprete para operar sobre ellos y surtir de esa línea de instrucciones a un intérprete. Parallel puede acelerar esto. Asumiendo que `archivo` contiene una lista de instrucciones del intérprete, una por línea,

```
parallel -j 10 < archivo
```

evaluará las instrucciones usando el intérprete (ya que no se ha proporcionado ninguna instrucción explícita como argumento), en bloques de diez tareas del intérprete cada vez.

3.3 Funciones del Intérprete

Las funciones del intérprete son una forma de agrupar instrucciones para su posterior ejecución usando un único nombre para el grupo. Son ejecutadas como una instrucción normal. Cuando se usa el nombre de una función del intérprete como un nombre de instrucción simple, se ejecuta la lista de instrucciones asociada con ese nombre de función. Las funciones del intérprete son ejecutadas en el contexto actual del intérprete; ningún nuevo proceso es creado para interpretarlas.

Las funciones se declaran usando esta sintaxis:

fname `()` *instrucción-compuesta* `[` *redirecciones* `]`

o

`function` *fname* `[()]` *instrucción-compuesta* `[` *redirecciones* `]`

Esto define una instrucción del intérprete llamada *fname*. La palabra reservada `function` es opcional. Si se proporciona la palabra reservada `function`, los paréntesis son opcionales. El *cuerpo* de la función es la instrucción compuesta *instrucción-compuesta* (véase Sección 3.2.5 [Instrucciones Compuestas], página 10). Esa instrucción es normalmente una *lista* entre { y }, pero puede ser cualquier instrucción compuesta listada anteriormente, con una excepción: si se usa la palabra reservada `function` pero no se proporcionan paréntesis, son obligatorias las llaves. *instrucción-compuesta* se ejecuta cada vez que se especifica *fname* como el nombre de una instrucción. Cuando el intérprete se encuentra en modo POSIX (véase Sección 6.11 [Modo POSIX de Bash], página 111, *fname* debe ser un *nombre* del intérprete válido y no puede ser el mismo que una de las instrucciones integradas especiales (véase Sección 4.4 [Instrucciones Integradas Especiales], página 80). En el modo predeterminado, un nombre de función puede ser cualquier palabra del intérprete no entrecomillada que no contenga un '`$`'. Todas las redirecciones (véase ⟨undefined⟩ [Redirections], página ⟨undefined⟩) asociadas con la función del intérprete se realizan cuando se ejecuta la función. Una definición de función puede ser eliminada usando la opción `-f` de la instrucción integrada `unset` (véase ⟨undefined⟩ [Instrucciones Integradas del Intérprete Bourne], página ⟨undefined⟩).

El estado de salida de una definición de función es cero a no ser que ocurra un error de sintaxis o una función de solo lectura con el mismo nombre ya exista. Cuando se ejecuta, el estado de salida de una función es el estado de salida de la última instrucción ejecutada en el cuerpo.

Tenga en cuenta que por razones históricas, en el modo más usual las llaves que rodean el cuerpo de la función tienen que estar separadas del cuerpo por `blanco`s o nuevas líneas. Esto es porque las llaves son palabras reservadas que solo son reconocidas como tales cuando están separadas de la lista de instrucciones por espacios en blanco u otros metacaracteres del intérprete. Además, al usar llaves, la *lista* debe terminar en un punto y coma, un '`&`' o una nueva línea.

Cuando se ejecuta una función, los argumentos para la función se convierten en los parámetros posicionales durante su ejecución (véase Sección 3.4.1 [Parámetros Posicionales], página 23). El parámetro especial '`#`' que se expande al número de parámetros posicionales se actualiza para reflejar el cambio. El parámetro especial `0` no es modificado. El primer elemento de la variable `FUNCNAME` es establecido al nombre de la función durante la ejecución de la función.

Todos los otros aspectos del entorno de ejecución del intérprete son idénticos entre una función y su ejecutor con estas excepciones: las traps `DEBUG` y `RETURN` no se heredan a no ser que se le haya dado a la función el atributo `trace` usando la instrucción integrada `declare` o la opción `-o functrace` haya sido habilitada con la instrucción integrada `set`, (en cuyo caso todas las funciones heredan las traps `DEBUG` y `RETURN`), y la trap `ERR` no se hereda a no ser que la opción del intérprete `-o errtrace` haya sido habilitada. Véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48, para la decripción de la instrucción integrada `trap`.

La variable `FUNCNEST`, si está establecida a un valor numérico mayor que 0, define el nivel máximo de anidamiento de funciones. Las llamadas de funciones que exceden el límite hacen que la instrucción completa se aborte.

Si la instrucción integrada `return` se ejecuta en una función, la función finaliza y la ejecución se reanuda con la siguiente instrucción después de la llamada a la función. Cualquier instrucción asociada con la trap `RETURN` es ejecutada antes de que la ejecución se reanude. Cuando una función finaliza, los valores de los parámetros posicionales y el parámetro especial '`#`' se restablecen a los valores que tenían antes de la ejecución de la función. Si se le pasa un argumento numérico a `return`, este es el estado de retorno de la función; si no, el estado de retorno de la función es el estado de salida de la última instrucción ejecutada antes de `return`.

Las variables locales a la función pueden ser declaradas con la instrucción integrada `local`. Estas variables son solo visibles para la función y las instrucciones que invoca. Esto es especialmente importante cuando una función del intérprete llama a otras funciones.

Las variables locales variables «sombra» con el mismo nombre declarado en diferentes alcances. Por ejemplo, una variable local declarada en una función esconde una variable global del mismo nombre: referencias y asignaciones se refieren a la variable local, dejando la variable global sin modificar. Cuando la función retorna, la variable global es visible de nuevo.

El intérprete usa el *alcance dinámico* para controlar la visibilidad de una variables dentro de funciones. Con el alcance dinámico, las variables visibles y sus valores son un resultado

de la secuencia de llamadas de función que hicieron que la ejecución llegara a la ubicación actual. El valor de una variable que una función ve depende de su valor dentro de su ejecutor, si existe, si el ejecutor es el alcance «global» u otra función del intérprete. Esto también es el valor a la que una declaración de variable local hace «sombra» y el valor que es restablecido cuando la función retorna.

Por ejemplo, si una variable *var* se declara como local en función *func1*, y *func1* llama a otra función *func2*, las referencias a *var* hechas desde dentro de *func2* resolverán a la variable local *var* de *func1*, haciendo sombra a cualquier variable global llamada *var*.

El siguiente guion demuestra este comportamiento. Al ser ejecutado, el guion muestra

```
In func2, var = func1 local
```

```
func1()
{
    local var='func1 local'
    func2
}

func2()
{
    echo "In func2, var = $var"
}

var=global
func1
```

La instrucción integrada `unset` también actúa usando el mismo alcance dinámico: si una variable es local al actual alcance, `unset` la eliminará; de lo contrario, la eliminación se referirá a la variable encontrada en cualquier alcance de llamada descrito arriba. Si una variable en el alcance local actual se elimina, permanecerá así hasta que sea restablecida en ese alcance o hasta que la función retorne. Una vez que la función retorna cualquier instancia de la variable en el alcance anterior se vuelve invisible. Si la eliminación actúa sobre una variable en un alcance anterior, cualquier instancia de una variable con ese nombre a la que se ha hecho sombra se volverá invisible.

Los nombres y definiciones de funciones pueden ser listados con la opción `-f` para la instrucción integrada `declare` (`typeset`) (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56). La opción `-F` para `declare` o `typeset` listará solo los nombres de funciones (y opcionalmente el archivo fuente y el número de línea, si la opción del intérprete `extdebug` está habilitada). Las funciones pueden ser importadas para que los subintérpretes las tengan definidas automáticamente con la opción `-f` para la instrucción integrada `export` (véase Capítulo 4 [Instrucciones Integradas del Intérprete], página 48).

Las funciones pueden ser recursivas. La variable `FUNCNEST` se puede usar para limitar la profundidad de la pila de llamadas y restringir el número de llamadas de función. Por defecto, no hay límite en el número de llamadas recursivas.

3.4 Parámetros del Intérprete

Un *parámetro* es una entidad que almacena valores. Puede ser un `nombre`, un número o uno de los caracteres especiales listados más abajo. Una *variable* es un parámetro denotado por

un `nombre`. Una variable tiene un *valor* y cero o más *atributos*. Los atributos son asignados usando la instrucción integrada `declare` (consulte la descripción de la instrucción integrada `declare` en Sección 4.2 [Instrucciones Integradas de Bash], página 56).

Un parámetro está establecido si se le ha asignado un valor. La cadena nula es un valor válido. Una vez que se asigne una variable, solo puede ser eliminada usando la instrucción integrada `unset`.

Se puede asignar a una variable con una declaración de la forma

nombre=[*valor*]

Si no se da *valor*, se asigna la cadena nula a la variable. Todos los *valor*es experimentan la expansión de virgulilla, la expansión de parámetros y de variables, sustitución de instrucciones, expansión aritmética y eliminación de comillas (detallada a continuación). Si la variable tiene establecido su atributo `integer`, entonces el *valor* es evaluado como una expansión aritmética incluso si no es usada la expansión `$((...))` (véase Sección 3.5.5 [Expansión Aritmética], página 34). No se realiza la separación de palabras, con la excepción de `"$@"` como se explica más abajo. No se realiza la expansión de nombre de archivo. Las declaraciones de asignaciones pueden aparecer también como argumentos para las instrucciones integradas `alias`, `declare`, `typeset`, `export`, `readonly` y `local` (instrucciones de *declaración*). Durante modo POSIX (véase Sección 6.11 [Modo POSIX de Bash], página 111), estas instrucciones integradas pueden aparecer en una instrucción después de una o más instancias de la instrucción integrada `command` y mantener estas propiedades de declaración de asignación.

En el contexto en el que una sentencia de asignación esté asignando un valor a una variable o índice de vector (véase Sección 6.7 [Vectores], página 105) del intérprete, el operador '`+=`' puede ser usado para añadir o sumar al valor previo de la variable. Esto incluye argumentos de instrucciones integradas como `declare` que aceptan sentencias de asignación (instrucciones de *declaración*). Cuando '`+=`' se aplica a una variable para la cual el atributo *integer* ha sido establecido, *valor* se evalúa como una expresión aritmética y se suma al valor actual de la variable, que también es evaluado. Cuando '`+=`' se aplica a una variable de vector usando la asignación compuesta (véase Sección 6.7 [Vectores], página 105), el valor de la variable no es eliminado (como lo es al usar '`=`'), y se adjuntan nuevos valores al vector empezando por un número a partir del índice máximo del vector (para vectores indexados), o se añade como una pareja clave-valor en un vector asociativo. Cuando se aplica a una variable con valor de cadena, *valor* se expande y se añade al valor de la variable.

Se puede asignar una variable el atributo *nameref* usando la opción `-n` para las instrucciones integradas `declare` o `local` (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56) para crear una *nameref*, o una referencia a otra variable. Esto permite que las variables sean manipuladas indirectamente. Cada vez que se haga referencia a la variable nameref, se asigne, elimine o sean modificados sus atributos (aparte de usando o cambiando el atributo nameref en sí), la operación se realiza en la variable especificada por el valor de la variable nameref. Un nameref se usa comúnmente dentro de las funciones del intérprete para referirse a una variable cuyo nombre es pasado como un argumento a la función. Por ejemplo, si un nombre de variable es pasado a una función del intérprete como su primer argumento, ejecutar

```
declare -n ref=$1
```

dentro de la función crea una variable nameref *ref* cuyo valor es el nombre de variable pasado como el primer argumento. Las referencias y asignaciones a *ref*, y cambios a sus atributos, son tratados como referencias, asignaciones y modificaciones de atributo a la variable cuyo nombre fue pasado a `$1`.

Si la variable de control en un bucle `for` tiene el atributo nameref, la lista de palabras puede ser una lista de variables del intérprete, y una referencia de nombre será establecida para cada palabra en la lista, en orden, cuando el bucle sea ejecutado. No se le puede dar el atributo nameref a las variables de vector. Sin embargo, las variables nameref pueden hacer referencia a variables de vector y variables de subíndice de vector. Los namerefs pueden ser eliminados usando la opción `-n` de la instrucción integrada `unset` (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48). De lo contrario, si se ejecuta `unset` con el nombre de una variable nameref como un argumento, la variable referenciada por la variable nameref será eliminada.

3.4.1 Parámetros Posicionales

Un *parámetro posicional* es un parámetro denotado por uno o más dígitos, aparte del único dígito `0`. Los argumentos posicionales se asignan desde los argumentos del intérprete cuando es llamado y pueden ser reasignados usando la instrucción integrada `set`. El parámetro posicional `N` puede ser referenciado como `${N}`, o como `$N` cuando `N` consta de un único dígito. Los parámetros posicionales no pueden ser asignados con declaraciones de asignación. Las instrucciones integradas `set` y `shift` son usadas para asignar y eliminarlos (véase Capítulo 4 [Instrucciones Integradas del Intérprete], página 48). Los parámetros posicionales son temporalmente reemplazados cuando se ejecuta una función del intérprete (véase Sección 3.3 [Funciones del Intérprete], página 19).

Cuando un parámetro posicional que consta de un único dígito se expande, debe estar rodeado de llaves.

3.4.2 Parámetros Especiales

El intérprete trata varios parámetros de forma especial. Estos parámetros solo pueden ser referenciados; las asignaciones a ellos no están permitidas.

`*` ($*) Se expande a los parámetros posicionales, empezando por uno. Cuando la expresión no está entre comillas dobles, cada parámetro posicional se expande a uno palabra separada. En contextos en los que se realiza, esas palabras están sujetas a una división de palabras adicional y a la expansión de nombre de archivo. Cuando la expansión ocurre dentro de comillas dobles, se expande a una sola palabra con el valor de cada parámetro separado por el primer carácter de la variable especial `IFS`. Es decir, `"$*"` equivale a `"$1`*c*`$2`*c*`..."`, donde *c* es el primer carácter del valor de la variable `IFS`. Si `IFS` no está asignada, los parámetros se separan por espacios. Si `IFS` es nula, los parámetros se unen sin separadores que intervengan.

`@` ($@) Se expande a los parámetros posicionales empezando por uno. En contextos donde se realiza la división de palabras, esto expande cada parámetro posicional a una palabra separada; si no está entre comillas dobles, estas palabras están sujetas a la división de palabras. En contextos donde no se realiza la división de palabras, esto se expande a una única palabra con cada parámetro

posicional separado por un espacio. Cuando la expansión ocurre dentro de comillas dobles y se realiza la división de palabras, cada parámetro se expande a una palabra separada. Es decir, `"$@"` es equivalente a `"$1" "$2" ...`. Si la expansión rodeada entre comillas dobles ocurre dentro de una palabra, la expansión del primer parámetro se una con la parte inicial de la palabra original y la expansión del último parámetro se una con la última parte de la palabra original. Cuando no hay parámetros posicionales, `"$@"` y `$@` se expanden a nada (es decir, son eliminados).

`#` ($#) Se expande al número de parámetros posicionales en decimal.

`?` ($?) Se expande al estado de salida de la tubería en primer plano ejecutada más recientemente.

`-` ($-, un guion.) Se expande a las actuales opciones centinelas como se especifica en la llamada, por la instrucción integrada `set` o aquellas establecidas por el propio intérprete (como la opción `-i`).

`$` ($$) Se expande al ID de proceso del intérprete. En un subintérprete `()`, se expande al ID de proceso del intérprete que lo llama, no del subintérprete.

`!` ($!) Se expande al ID de proceso de tarea ubicada en segundo plano más recientemente, bien ejecutada como una instrucción asíncrona o usando la instrucción integrada `bg` (véase Sección 7.2 [Instrucciones Integradas de Control de Tareas], página 119).

`0` ($0) Se expande al nombre del intérprete o el guion del intérprete. Esto es establecido durante la inicialización del intérprete. Si Bash se invoca con un archivo de instrucciones (véase Sección 3.8 [Guiones del Intérprete], página 46), `$0` se establece al nombre de ese archivo. Si Bash se inicia con la opción `-c` (véase Sección 6.1 [Llamando a Bash], página 95), entonces `$0` se establece al primer argumento después de que la cadena sea ejecutada, si una está presente. De lo contrario, es establecido al nombre de archivo usado para llamar a Bash, como si fuera dado como argumento cero.

3.5 Expansiones del Intérprete

La expansión se realiza en la línea de órdenes después de que haya sido dividida en `símbolo`s. Hay siete tipos de expansión realizados:

- expansión de llaves
- expansión de virgulilla
- expansión de parámetros y variables
- sustitución de instrucciones
- expansión aritmética
- división de palabras
- expansión de nombre de archivo

El orden de las expansiones es expansión de llaves; expansión de virgulilla, expansión de parámetros y variables, expansión aritmética y sustitución de instrucciones (hechas de izquierda a derecha); división de palabras; y expansión de nombre de archivo.

En sistemas que lo soportan, hay un expansión adicional disponible: *sustitución de proceso*. Esta se realiza al mismo tiempo que la expansión de virgulilla, parámetros y variables y la sustitución de instrucciones.

Después de que se hayan realizado estas expansiones, los caracteres de entrecomillado presentes en la palabra original son eliminados a no ser que estos mismos hayan sido entrecomillados (*eliminación de comillas*).

Solo la expansión de llaves, división de palabras y la expansión de nombre de archivo pueden aumentar el número de palabras de la expansión; otras expansiones expanden una sola palabra a una sola palabra. Las únicas excepciones a esto son las expansiones de `"$@"` y `$*` (véase Sección 3.4.2 [Parámetros Especiales], página 23, y `${nombre[*]}` (véase Sección 6.7 [Vectores], página 105).

Después de todas las expansiones, se realiza la `eliminación de comillas` (véase Sección 3.5.9 [Eliminación de Comillas], página 37).

3.5.1 Expansión de Llaves

La expansión de llaves es un mecanismo por el cual se pueden generar cadenas arbitrarias. Este mecanismo es similar a la *expansión de nombre de archivo* (véase Sección 3.5.8 [Expansión de Nombre de Archivo], página 35), pero los nombres de archivo generados no necesitan existir. Los patrones que serán expandidos por llaves toman la forma de un *preámbulo* opcional, seguido de una serie de cadenas separadas por comas o una expresión secuencial entre un par de llaves, seguida por un *epílogo* opcional. El preámbulo es prefijado a cada cadena contenida entre las llaves, y el epílogo es entonces añadido a cada cadena resultante, expandiendo de izquierda a derecha.

Las expansiones de llaves pueden ser anidadas. Los resultados de cada cadena expandida no se ordenan; se preserva el orden de izquierda a derecha. Por ejemplo,

```
bash$ echo a{d,c,b}e
ade ace abe
```

Una expresión secuencial toma la forma `{x..y[..incr]}`, donde *x* e *y* son enteros o caracteres únicos, y *incr*, un incremento opcional, es un entero. Cuando se proporcionan enteros, la expresión se expande a cada número entre *x* y *y*, inclusive. Los enteros proporcionados pueden ser prefijados con '`0`' para forzar que cada término tenga la misma anchura. Cuando *x* o *y* comienzan en cero, el intérprete trata de obligar a todos los términos generados contener el mismo número de dígitos, rellenando con ceros donde sea necesario. Cuando se proporcionan los caracteres, la expresión se expande a cada carácter lexicográficamente entre *x* e *y*, inclusive, usando la configuración regional predeterminada C. Observe que tanto *x* e *y* deben ser del mismo tipo. Cuando se proporciona el incremento, se usa como diferencia entre cada término. El incremento predeterminado es 1 o -1 según corresponda.

La expansión de llaves se realiza antes que cualquier otra expansión, y se preservan todos los caracteres especiales de otras expansiones en el resultado. Es estrictamente textual. Bash no aplica ninguna interpretación sintáctica al contexto de la expansión o al texto entre las llaves.

Una expansión de llaves correctamente formada debe contener llaves de apertura y de cierre sin entrecomillar, y al menos una coma sin entrecomillar o una expresión secuencial válida. Cualquier expansión de llaves formada incorrectamente se deja sin modificar.

Un { o ‘,’ puede entrecomillarse con una barra invertida para evitar que sea considerado parte de la expresión de llaves. Para evitar conflictos con la expansión de parámetros, la cadena ‘${’ no se considera admisible para la expansión de llaves e inhibe la expansión de llaves hasta el ‘}’ de cierre.

Esta construcción se usa típicamente como atajo cuando el prefijo común de las cadenas que van a ser generadas es más largo que en el siguiente ejemplo:

```
mkdir /usr/local/src/bash/{old,new,dist,bugs}
```

o

```
chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}
```

3.5.2 Expansión de Virgulilla

Si una palabra comienza por un carácter de virgulilla sin entrecomillar (‘~’), todos los caracteres hasta la primera barra sin entrecomillar (o todos los caracteres, si no hay barra sin entrecomillar) son considerados un *prefijo de virgulilla*. Si ninguno de los caracteres en el prefijo de virgulilla está entrecomillado, los caracteres en el prefijo de virgulilla que siguen a la virgulilla son tratados como un posible *nombre de acceso*. Si este nombre de acceso es la cadena nula, la virgulilla se reemplaza por el valor de la variable `HOME` del intérprete. Si `HOME` no está establecida, se sustituye el directorio personal del usuario ejecutando el intérprete en su lugar. De lo contrario, se reemplaza el prefijo de virgulilla por el directorio personal asociado con el nombre de acceso especificado.

Si el prefijo de virgulilla es ‘~+’, el valor de la variable del intérprete `PWD` reemplaza el prefijo de virgulilla. Si el prefijo de virgulilla es ‘~-’, se sustituye el valor de la variable `OLDPWD`, si está establecida.

Si los caracteres que siguen la virgulilla en el prefijo de virgulilla constan de un número *N*, opcionalmente prefijado por un ‘+’ o un ‘-’, la expansión de virgulilla se reemplaza por el elemento correspondiente de la pila de directorios, como se mostraría por la instrucción integrada `dirs` llamada con los caracteres que siguen a la virgulilla en el prefijo de virgulilla como un argumento (véase Sección 6.8 [La Pila de Directorios], página 107). Si el prefijo de virgulilla, sin la virgulilla, consta de un número sin un ‘+’ o ‘-’ inicial, se asume ‘+’.

Si el nombre de acceso es inválido, o la expansión de virgulilla falla, la palabra se deja sin modificar.

Cada asignación de variable se revisa en busca de prefijos de virgulilla sin entrecomillar que inmediatamente sigan a un ‘:’ o al primer ‘=’. En estos casos, también es realizada la expansión de virgulilla. Consecuentemente, uno puede usar nombres de archivo con virgulillas en asignaciones a `PATH`, `MAILPATH` y `CDPATH`, y el intérprete asigna el valor expandido.

La siguiente tabla muestra cómo trata Bash los prefijos de virgulilla sin entrecomillar:

`~` El valor de `$HOME`

`~/foo` `$HOME/foo`

`~fred/foo`
El subdirectorio `foo` del directorio personal del usuario `fred`

`~+/foo` `$PWD/foo`

`~-/foo` `${OLDPWD-'~-'}/foo`

`~N` La cadena que sería mostrada por '`dirs +N`'

`~+N` La cadena que sería mostrada por '`dirs +N`'

`~-N` La cadena que sería mostrada por '`dirs -N`'

Bash también realiza la expansión de virgulilla en palabras que satisfacen las condiciones de asignación de variables (véase Sección 3.4 [Parámetros del Intérprete], página 21) cuando aparecen como argumentos de instrucciones simples. Bash no hace esto, aparte de para las instrucciones de *declaración* listadas arriaba, en el modo POSIX

3.5.3 Expansión de Parámetros del Intérprete

El carácter '`$`' introduce la expansión de parámetros, sustitución de instrucciones o expansión aritmética. El nombre o símbolo del parámetro para ser expandido se puede encerrar en llaves, que son opcionales pero sirven para evitar que la variable se expanda de caracteres que inmediatamente lo siguen que podrían ser interpretados como parte del nombre.

Cuando se usan llaves, la llave de cierre coincidente es el primer '`}`' no escapado por una barra invertida o dentro de una cadena entrecomillada, y no dentro de una expresión aritmética, sustitución de instrucción o expansión de parámetro incrustadas.

La forma básica de expansión de parámetros es ${*parámetro*}. El valor de *parámetro* es sustituido. El *parámetro* es un parámetro del intérprete como se describe arriba (véase Sección 3.4 [Parámetros del Intérprete], página 21) o una referencia de vector (véase Sección 6.7 [Vectores], página 105). Las llaves son requeridas cuando *parámetro* es un parámetro posicional con más de un dígito, o cuando *parámetro* es seguido por un carácter que no está pensado para ser interpretado como parte de su nombre.

Si el primer carácter de *parámetro* es un signo de exclamación (!) y *parámetro* no es un *nameref*, introduce un nivel de indirección. Bash usa el valor formado al expandir el resto de *parámetro* como el nuevo *parámetro*; este es luego expandido y ese valor se usa en el resto de la expansión, en vez de la expansión del *parámetro* original. Esto se conoce como `expansión indirecta`. El valor está sujeto a la expansión de virgulilla, la expansión de parámetros, la sustitución de instrucciones y la expansión aritmética. Si *parámetro* es un nameref, esto se expande al nombre de la variable referenciada por *parámetro* en vez de realizar la expansión indirecta completa. Las excepciones a esto son las expansiones de ${!*prefijo**} y ${!*nombre*[@]} descritas abajo. El signo de exclamación debe seguir inmediatamente a la llave izquierda para introducir la indirección.

En todos los siguientes casos, *palabra* está sujeta a la expansión de virgulilla, expansión de parámetro, sustitución de instrucción y expansión aritmética.

Cuando no se realiza la expansión de subcadena, usando la forma descrita abajo (p. ej., '`:-`'), Bash comprueba por un parámetro que sea nulo o no esté establecido. Omitir los dos puntos resulta en una prueba solo para un parámetro que no esté establecido. Dicho de otra manera, si se incluyen los dos puntos, el operador comprueba la existencia de *parámetro* y que su valor no sea nulo; si se omiten los dos puntos, el operador solo comprueba su existencia.

`${parámetro:-palabra}`

Si *parámetro* no está establecido o es nulo, se sustituye por la expansión de *palabra*. De lo contrario, se sustituye por el valor de *parámetro*.

`${parámetro:=palabra}`
: Si *parámetro* no está establecido o es nulo, se asigna la expansión de *palabra* a *parámetro*. Se sustituye entonces el valor de *parámetro*. A los parámetros posicionales y parámetros especiales no se les puede asignar de esta manera.

`${parámetro:?palabra}`
: Si *parámetro* no está establecido o es nulo, la expansión de *palabra* (o un mensaje al respecto si *palabra* no está presente) es escrita al error estándar, y el intérprete, si no es interactivo, se cierra. De lo contrario, se sustituye el valor de *parámetro*.

`${parámetro:+palabra}`
: Si *parámetro* es nulo o no está establecido, no se sustituye por nada; de lo contrario, se sustituye por la expansión de *palabra*.

`${parámetro:desplazamiento}`
`${parámetro:desplazamiento:longitud}`
: Esto se conoce como Expansión de Subcadena. Se expande hasta *longitud* carácteres del valor de *parámetro* empezando por el carácter especificado por *desplazamiento*. Si *parámetro* es '`@`', un vector indexado subindexado por '`@`' o '`*`' o un nombre de vector asociativo, los resultados difieren como se describe más abajo. Si se omite *longitud*, se expande a la subcadena del valor de *parámetro* empezando en el carácter especificado por *desplazamiento* y extendiéndose hasta el final del valor. *longitud* y *desplazamiento* son expresiones aritméticas (véase Sección 6.5 [Aritmética del Intérprete], página 103).

  Si *desplazamiento* se evalúa a un número menor que cero, el valor se usa como un desplazamiento en caracteres desde el final del valor de *parámetro*. Si *longitud* se evalúa a un número menor que cero, se interpreta como un desplazamiento en caracteres desde el final del valor de *parámetro* en vez de un número de caracteres, y la expansión equivale a los caracteres entre *desplazamiento* y ese resultado. Observe que un desplazamiento negativo debe ser separado de los dos puntos por al menos un espacio para evitar confusión con la expansión '`:-`'.

  Aquí hay algunos ejemplos que ilustran la expansión de subcadenas en parámetros y vectores subindexados:

```
$ cadena=01234567890abcdefgh
$ echo ${cadena:7}
7890abcdefgh
$ echo ${cadena:7:0}

$ echo ${cadena:7:2}
78
$ echo ${cadena:7:-2}
7890abcdef
$ echo ${cadena: -7}
bcdefgh
$ echo ${cadena: -7:0}

$ echo ${cadena: -7:2}
```

```
bc
$ echo ${cadena: -7:-2}
bcdef
$ set -- 01234567890abcdefgh
$ echo ${1:7}
7890abcdefgh
$ echo ${1:7:0}

$ echo ${1:7:2}
78
$ echo ${1:7:-2}
7890abcdef
$ echo ${1: -7}
bcdefgh
$ echo ${1: -7:0}

$ echo ${1: -7:2}
bc
$ echo ${1: -7:-2}
bcdef
$ vector[0]=01234567890abcdefgh
$ echo ${vector[0]:7}
7890abcdefgh
$ echo ${vector[0]:7:0}

$ echo ${vector[0]:7:2}
78
$ echo ${vector[0]:7:-2}
7890abcdef
$ echo ${vector[0]: -7}
bcdefgh
$ echo ${vector[0]: -7:0}

$ echo ${vector[0]: -7:2}
bc
$ echo ${vector[0]: -7:-2}
bcdef
```

Si *parámetro* es '`@`', el resultado es *longitud* parámetros posicionales comenzando por *desplazamiento*. Un *desplazamiento* se toma en relación a uno mayor que el parámetro posicional más grande, de manera que un desplazamiento de -1 evalúa al último parámetro posicional. Es un error de expansión si *longitud* evalúa a un número menor que cero.

Los siguientes ejemplos ilustran la expansión de subcadenas usando parámetros posicionales:

```
$ set -- 1 2 3 4 5 6 7 8 9 0 a b c d e f g h
$ echo ${@:7}
```

```
7 8 9 0 a b c d e f g h
$ echo ${@:7:0}

$ echo ${@:7:2}
7 8
$ echo ${@:7:-2}
bash: -2: expresi@'on de subcadena < 0
$ echo ${@: -7:2}
b c
$ echo ${@:0}
./bash 1 2 3 4 5 6 7 8 9 0 a b c d e f g h
$ echo ${@:0:2}
./bash 1
$ echo ${@: -7:0}
```

Si *parámetro* es un nombre de vector subindexado por '`@`' o '`*`', el resultado es la *longitud* de miembros del vector que comienzan por `${`*parámetro*`[`*desplazamiento*`]}`. Un *desplazamiento* negativo se toma respecto a uno mayor que el máximo índice del vector especificado. Es un error de expansión si *longitud* evalúa a un número menor que cero.

Estos ejemplos muestran cómo puede usar la expansión de subcadenas con vectores indexados:

```
$ vector=(0 1 2 3 4 5 6 7 8 9 0 a b c d e f g h)
$ echo ${vector[@]:7}
7 8 9 0 a b c d e f g h
$ echo ${vector[@]:7:2}
7 8
$ echo ${vector[@]: -7:2}
b c
$ echo ${vector[@]: -7:-2}
bash: -2: expresi@'on de subcadena < 0
$ echo ${vector[@]:0}
0 1 2 3 4 5 6 7 8 9 0 a b c d e f g h
$ echo ${vector[@]:0:2}
0 1
$ echo ${vector[@]: -7:0}
```

La expansión de subcadena aplicada a un vector asociativo produce resultados inesperados.

El indexado de subcadena está basado en ceros a no ser que se usen los parámetros posicionales, en cuyo caso el indexado comienza por defecto por 1. Si *desplazamiento* es 0 y se usan los parámetros posicionales, se prefija `$0` a la lista.

`${!`*prefijo*`*}`
`${!`*prefijo*`@}`
: Se expande a los nombres de variables cuyos nombres empiezan por *prefijo*, separados por el primer carácter de la variable especial `IFS`. Cuando se usa '`@`' y la expansión aparece entre comillas dobles, cada nombre de variable se expande a una palabra separada.

`${!`*nombre*`[@]}`
`${!`*nombre*`[*]}`
: Si *nombre* es una variable de vector, se expande a la lista de indices de vectores (claves) asignada en *nombre*. Si *nombre* no es un vector, se expande a 0 si *nombre* está establecido y a nulo en caso contrario. Cuando se usa '`@`' y la expansión aparece entre comillas dobles, cada clave se expande a una palabra separada.

`${#`*parámetro*`}`
: Se sustituye la longitud en caracteres del valor expandido de *parámetro*. Si *parámetro* es '`*`' o '`@`', el valor sustituido es el número de parámetros posicionales. Si *parámetro* es un nombre de vector subindexado por '`*`' o '`@`', el valor sustituido es el número de elementos en el vector. Si *parámetro* es un nombre de vector indexado subindexado por un número negativo, ese número es interpretado como relativo a uno mayor que el índice máximo de *parámetro*, de forma que los índices negativos cuenten atrás desde el final del vector, y un índice de -1 referencie al último elemento.

`${`*parámetro*`#`*palabra*`}`
`${`*parámetro*`##`*palabra*`}`
: La *palabra* se expande para producir un patrón y es coincidida según las reglas descritas abajo (véase Sección 3.5.8.1 [Coincidencia de Patrones], página 36). Si el patrón coincide con el principio del valor expandido de *parámetro*, el resultado de la expansión es el valor expandido de *parámetro* con el patrón coincidente más corto (el caso '`#`') o el patrón coincidente más largo (el caso '`##`'). Si *parámetro* es '`@`' o '`*`', se aplica la operación de eliminación de patrón a cada parámetro posicional en orden, y la expansión es la lista resultante. Si *parámetro* es una variable de vector subindexada con '`@`' o '`*`', la operación de eliminación de patrón se aplica a cada miembro del vector en orden, y la expansión es la lista resultante.

`${`*parámetro*`%`*palabra*`}`
`${`*parámetro*`%%`*palabra*`}`
: La *palabra* se expande para producir un patrón y es coincidida según las reglas descritas a continuación (véase Sección 3.5.8.1 [Coincidencia de Patrones], página 36). Si el patrón coincide con una porción final del valor expandido de *parámetro*, entonces el resultado de la expansión es el valor de *parámetro* con el patrón más corto que coincide (el `case` '`%`') o el patrón más largo que coincide (el `case` '`%%`') eliminado. Si *parámetro* es '`@`' o '`*`', la operación de eliminación del patrón se aplica a cada parámetro posicional en orden y la expansión es la lista resultante. Si *parámetro* es un vector suscrito con '`@`' o '`*`', la operación

de eliminación del patrón se aplica a cada miembro del vector en orden y la expansión es la lista resultante.

`${`*parámetro*`/`*patrón*`/`*cadena*`}`
: El *patrón* se expande para producir un patrón de la misma manera que en la expansión de nombre de archivo. Se expande *parámetro*, y se reemplaza por *cadena* la coincidencia de *patrón* con el valor más largo. Si *patrón* comienza por '`/`', todas las coincidencias de *patrón* son reemplazadas por *cadena*. La coincidencia se realiza según las reglas descritas abajo (véase Sección 3.5.8.1 [Coincidencia de Patrones], página 36). Normalmente solo se reemplaza la primera coincidencia. Si *patrón* comienza en '`#`', debe coincidir al comienzo del valor expandido de *parámetro*. Si *patrón* comienza por '`%`', debe coincidir al final del valor expandido de *parámetro*. Si *cadena* es nulo, las coincidencias de *patrón* son eliminadas y se puede omitir el `/` que sigue a *patrón*. Si está habilitada la opción del intérprete `nocasematch` (consulte la descripción de `shopt` en Sección 4.3.2 [La Instrucción Integrada Shopt], página 73), la coincidencia se realiza sin importar las mayúsculas y minúsculas de caracteres alfabéticos. Si *parámetro* es '`@`' o '`*`', la operación de sustitución se aplica a cada parámetro posicional en orden y la expansión es la lista resultante. Si *parámetro* es una variable de vector subindexada con '`@`' o '`*`', se aplica la operación de sustitución a cada miembro del vector en orden y la expansión es la lista resultante.

`${`*parámetro*`^`*patrón*`}`
`${`*parámetro*`^^`*patrón*`}`
`${`*parámetro*`,`*patrón*`}`
`${`*parámetro*`,,`*patrón*`}`
: La expansión modifica las mayúsculas y minúsculas de los caracteres alfabéticos. El *patrón* se expande para producir un patrón de la misma manera que en la expansión de nombre de archivo. Cada carácter en el valor expandido de *parámetro* es comprobado contra *patrón*, y, si coincide con el patrón, se alterna entre mayúsculas y minúsculas. El patrón no debería tratar de coincidir con más de un carácter. El operador '`^`' convierte letras minúsculas que coinciden con *patrón* a mayúsculas; el operador '`,`' convierte letras mayúsculas coincidentes a minúsculas. Las expansiones '`^^`' y '`,,`' convierten cada carácter coincidente en el valor expandido; las expansiones '`^`' y '`,`' coinciden y convierten solo el primer carácter en el valor expandido. Si se omite *patrón*, se trata como un '`?`', que coincide con cualquier carácter. Si *parámetro* es '`@`' o '`*`', la operación de modificación de alternación entre mayúsculas y minúsculas se aplica a cada parámetro posicional en orden y la expansión es la lista resultante. Si *parámetro* es una variable de vector subindexada con '`@`' o '`*`', la operación de modificación de alternación entre mayúsculas y minúsculas se aplica a cada miembro en orden y la expansión es la lista resultante.

`${`*parámetro*`@`*operador*`}`
: La expansión es o una transformación del valor de *parámetro* o información sobre *parámetro* en sí, dependiendo del valor de *operador*. Cada *operador* es una única letra:

`U` La expansión es una cadena que es el valor de *parámetro* en caracteres alfabéticos en minúscula convertidos a mayúscula.

`u` La expansión es una cadena que es el valor de *parámetro* con el primer carácter convertido a mayúscula, si es alfabético.

`L` La expansión es una cadena que es el valor de *parámetro* con los caracteres alfabéticos en mayúscula convertidos a minúscula.

`Q` La expansión es una cadena que es el valor de *parámetro* entrecomillado en un formato que puede ser reutilizado como entrada.

`E` La expansión es una cadena que es el valor de *parámetro* con secuencias de escape de barras bajas expandidas con el mecanismo de entrecomillado `$'...'`.

`P` La expansión es una cadena que es el resultado de expandir el valor de *parámetro* como si fuera una cadena de prompt (véase Sección 6.9 [Controlando el Prompt], página 109).

`A` La expansión es una cadena en forma de una sentencia de asignación o instrucción `declare` que, si se evalúa, recreará *parámetro* con sus atributos y valor.

`K` Produce una versión probablemente entrecomillada del valor de *parámetro*, a excepción de que muestra los valores de los vectores indexados y asociativos como una secuencia de parejas entrecomillados de clave-valor. (véase Sección 6.7 [Vectores], página 105).

`a` La expansión es una cadena consistente en valores centinela que representan los atributos de *parámetro*.

Si *parámetro* es '`@`' o '`*`', se aplica la operación a cada parámetro posicional en orden y la expansión es la lista resultante. Si *parámetro* es una variable de vector subindexada con '`@`' o '`*`', la operación se aplica a cada miembro del vector en orden y la expansión es la lista resultante.

El resultado de la expansión está sujeto a la separación de palabras y a la expansión de nombre de archivo descritas a continuación.

3.5.4 Sustitución de Instrucciones

La sustitución de instrucciones permite a la salida de una instrucción reemplazar a la propia instrucción. La sustitución de instrucción ocurre cuando se rodea una instrucción así:

```
$(instrucción)
```

o

```
`instrucción`
```

Bash realiza la expansión ejecutando *instrucción* en un entorno de subintérprete y reemplazando la sustitución de la instrucción con la salida estándar de la instrucción, con cualquier nueva línea final eliminada. Las nuevas líneas integradas no se eliminan, pero pueden ser borradas durante la división de palabras. La instrucción de sustitución `$(cat file)` puede ser reemplazada por el equivalente pero más rápido `$(< archivo)`.

Cuando se usa la vieja forma de sustitución de comilla invertida, la barra invertida mantiene su significado literal excepto cuando es seguida por '`$`', '`` ` ``' o '`\`'. La primera barra invertida no precedida por una barra invertida termina la sustitución de instrucción. Al usar la forma `$(`*`instrucción`*`)`, todos los caracteres entre los paréntesis componen la instrucción; ninguno es tratado de forma especial.

Las sustituciones de instrucciones se pueden anidar. Para anidar usando la forma de comilla invertida, escape las comillas invertidas interiores con barras invertidas.

Si la sustitución aparece entre comillas dobles, la división de palabras y la expansión de nombre de archivo no se realizan en los resultados.

3.5.5 Expansión Aritmética

La expansión aritmética permite la evaluación de una expresión aritmética y la sustitución del resultado. El formato para la expansión aritmética es:

```
$(( expresión ))
```

La expresión se trata como si estuviera entre comillas dobles, pero una comilla doble dentro de los paréntesis no se trata de forma especial. Todos los símbolos en la expresión pasan por la expansión de parámetro y variable, sustitución de instrucción y eliminación de comillas. El resultado se trata como la expresión aritmética a evaluar. Las expansiones aritméticas pueden ser anidadas.

La evaluación se realiza de acuerdo a las reglas listadas abajo (véase Sección 6.5 [Aritmética del Intérprete], página 103). Si la expresión es inválida, Bash imprime un mensaje indicando fallo al error estándar y no ocurre ninguna sustitución.

3.5.6 Sustitución de Procesos

La sustitución de proceso permite hacer referencia a la entrada o salida de un proceso usando un nombre de archivo. Toma la forma de

```
<(lista)
```

o

```
>(lista)
```

La *lista* de proceso es ejecutada asíncronamente, y su entrada o salida aparece como un nombre de archivo. Este nombre de archivo es pasado como un argumento a la instrucción actual como el resultado de la expansión. Si se usa la forma `>(`*`lista`*`)`, el archivo pasado como un argumento debería ser leído para obtener la salida de *lista*. Observe que no puede aparecer ningún espacio entre `<` o `>` y el paréntesis izquierdo, de lo contrario la construcción sería interpretada como una redirección. La sustitución de procesos está soportada en sistemas que admiten tuberías nombradas (FIFOs) o el método `/dev/fd` de nombrar archivos abiertos.

Cuando está disponible, la sustitución de proceso se realiza simultáneamente con la expansión de parámetro y de variable, la sustitución de instrucción y la expansión aritmética.

3.5.7 División de Palabras

El intérprete escanea los resultados de la expansión de parámetros, la sustitución de instrucción y la expansión aritmética que no ocurrieron dentro de comillas dobles para división de palabras.

El intérprete trata cada carácter de `$IFS` como un delimitador y divide los resultados de las otras expansiones en palabras usando estos caracteres como terminadores de campos. Si `IFS` no está habilitado o su valor es exactamente `<espacio><tabulación><nueva línea>`, el predeterminado, las secuencias de `<espacio>`, `<tabulación>` y `<nueva línea>` al principio y el final de los resultados de las expansiones previas son ignorados, y cualquier secuencia de caracteres `IFS` que no estén al principio o al final sirve para delimitar palabras. Si `IFS` tiene un valor distinto al predeterminado, las secuencias de caracteres de espacios en blanco `espacio`, `tabulación` y `nueva línea` son ignoradas al principio y final de la palabra, mientras que el carácter de espacio en blanco esté en el valor de `IFS` (un carácter en blanco de `IFS`). Cualquier carácter en `IFS` que no sea un espacio en blanco de `IFS`, junto a algún carácter de espacio en blanco de `IFS`, delimita un campo. Una secuencia de caracteres de espacio en blanco de `IFS` también se trata como un delimitador. Si el valor de `IFS` es nulo, no ocurre ninguna división de palabra.

Los argumentos explícitamente nulos (`""` o `''`) son conservados y pasados a instrucciones como cadenas vacías. Los argumentos nulos implícitos, resultantes de la expansión de parámetros que no tiene valores, son eliminados. Si un parámetro sin valor se expande entre comillas dobles, resulta en un argumento nulo y es retenido y pasado a una instrucción como una cadena vacía. Cuando un argumento nulo entrecomillado aparece como parte de una palabra cuya expansión no es nula, se elimina el argumento nulo. Es decir, la palabra `-d"` se convierte en `-d` después de la división de palabras y la eliminación de argumentos nulos.

Observe que si no ocurre ninguna expansión, no se realiza ninguna división.

3.5.8 Expansión de Nombre de Archivo

Después de la separación de palabras, a no ser que se especifique la opción `-f` (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68), Bash busca en cada palabra los caracteres '`*`', '`?`' y '`|`'. Si aparece alguno de estos caracteres, entonces la palabra se trata como un *patrón* y se reemplaza con una lista de nombres de archivo ordenada alfabéticamente que coincide con el patrón (véase Sección 3.5.8.1 [Coincidencia de Patrones], página 36). Si no se encuentran nombres de archivo que coincidan y la opción `nullglob` está deshabilitada, la palabra permanece como está. Si está activada la opción `nullglob` y no se encuentran coincidencias, se elimina la palabra. Si la opción `failglob` está activada y no se encuentran coincidencias, se muestra un mensaje de error y no se ejecuta la instrucción. Si está activada la opción del intérprete `nocaseglob`, entonces la comprobación se realiza sin tener en cuenta si los caracteres están en mayúscula o minúscula.

Cuando se usa un patrón para la expansión de nombre de archivo, el carácter '`.`' al principio de un nombre de archivo o inmediatamente siguiendo a una barra debe ser coincidido explícitamente, a no ser que la opción del intérprete `dotglob` esté habilitada. Los nombres de archivo '`.`' y '`..`' deben ser coincididos siempre explícitamente, incluso si `dotglob` está habilitada. En otros casos, el carácter '`.`' no es tratado de forma especial.

Al coincidir con un nombre de archivo, el carácter de barra debe ser siempre coincidido explícitamente por una barra en el patrón, pero en otros contextos de coincidencia puede ser coincidido por un carácter de patrón especial como se describe abajo (véase Sección 3.5.8.1 [Coincidencia de Patrones], página 36).

Consulte la descripción de `shopt` en Sección 4.3.2 [La Instrucción Integrada Shopt], página 73, para una descripción de las opciones `nocaseglob`, `nullglob`, `failglob` y `dotglob`.

La variable del intérprete `GLOBIGNORE` puede ser usada para restringir el conjunto de nombres de archivo que coinciden con un patrón. Si `GLOBIGNORE` está establecida, cada nombre de archivo coincidente que también coincida con uno de los patrones en `GLOBIGNORE` es eliminado de la lista de coincidencias. Si la opción `nocaseglob` está habilitada, la coincidencia con el patrón en `GLOBIGNORE` se realiza sin importar las mayúsculas y minúsculas. Los nombres de archivo `.` y `..` siempre se ignoran cuando `GLOBIGNORE` está establecida y no es nula. Sin embargo, establecer `GLOBIGNORE` a un valor no nulo tiene el efecto de activar la opción del intérprete `dotglob`, para que todos los otros nombres de archivo que empiezan por '`.`' coincidan. Para obtener el viejo comportamiento de ignorar los nombres de archivo que comienzan por '`.`', haz '`.*`' uno de los patrones en `GLOBIGNORE`. La opción `dotglob` se desabilita cuando `GLOBIGNORE` no está establecida.

3.5.8.1 Coincidencia de Patrones

Cualquier carácter que aparezca en un patrón, distinto a los caracteres de patrón especial descritos más abajo, coincide con sí mismo. El carácter NUL no puede ocurrir en un patrón. Una barra invertida escapa el siguiente carácter; la barra invertida de escape se descarta durante la coincidencia. Los caracteres de patrones especiales deben ser entrecomillados si no van a ser coincididos literalmente.

Los caracteres de patrón especiales tienen los siguientes significados:

`*` Coincide con cualquier cadena, incluida la cadena nula. Cuando está habilitada la opción `globstar` y se usa '`*`' en un contexto de expansión de nombre de archivo, dos '`*`' adyacentes usados como un patrón único coincidirán con todos los archivos y cero o más directorios y subdirectorios. Detrás de una '`/`', dos '`*`' adyacentes solo coincidirán con directorios y subdirectorios.

`?` Coincide con un único carácter.

`[...]` Coincide con cualquiera de los patrones incluidos. Una pareja de patrones separada por un guion denota una *expresión de rango*; cualquier carácter que se encuentre entre esos dos caracteres, inclusive, usando la secuencia de ordenación y el conjunto de caracteres de la configuración regional actual, es coincidido. Si el primer carácter que sigue al '`[`' es un '`!`' o un '`^`', es coincidido cualquier carácter no incluido. Un '−' puede ser coincidido incluyéndolo como el primer o último carácter en el conjunto. Un '`]`' puede ser coincidido incluyéndolo como el primer carácter en el conjunto. El orden de ordenación de caracteres en las expresiones de rango se determina por la configuración regional actual y los valores de las variables del intérprete `LC_COLLATE` y `LC_ALL`, si están establecidas.

Por ejemplo, en la configuración regional predeterminada C, '`[a-dx-z]`' es equivalente a '`[abcdxyz]`'. Muchas configuraciones regionales ordenan los caracteres en orden de diccionario, y en estas configuraciones regionales '`[a-dx-z]`' no es normalmente equivalente a '`[abcdxyz]`'; podría ser equivalente a '`[aBbCcDdxXyYz]`', por ejemplo. Para obtener la interpretación tradicional de rangos en las expresiones de llaves, puede forzar el uso de la

configuración regional C estableciendo la variable de entorno `LC_COLLATE` o `LC_ALL` al valor '`C`', o habilitar la opción del intérprete `globasciiranges`.

Dentro de '`[`' y '`]`', se pueden especificar *clases de caracteres* usando la sintaxis `[:`*clase*`:]`, donde *clase* es una de las siguientes clases definidas en el estándar POSIX:

```
alnum   alpha   ascii   blank   cntrl   digit   graph   lower
print   punct   space   upper   word    xdigit
```

Una clase de carácter coincide cualquier carácter que perteneciente a esa clase. La clase de carácter `word` coincide con letras, dígitos y el carácter '`_`'.

Dentro de '`[`' y '`]`', se puede especificar una *clase de equivalencia* usando la sintaxis `[=`*c*`=]`, que coincide con todos los caracteres con el mismo peso de ordenación (según esté definido en la configuración regional actual) como el carácter *c*.

Dentro de '`[`' y '`]`', la sintaxis `[.`*símbolo*`.]` coincide con el símbolo de ordenación *símbolo*.

Si la opción del intérprete `extglob` se habilita usando la instrucción integrada `shopt`, se reconocen varios operadores de coincidencia de patrón extendido. En la siguiente descripción, una *lista-de-patrones* es una lista de uno o más patrones separados por un '`|`'. Se pueden formar patrones compuestos usando uno o más de los siguientes subpatrones:

`?(`*lista-de-patrones*`)`
: Coincide con cero o una ocurrencia de los patrones dados.

`*(`*lista-de-patrones*`)`
: Coincide con cero o más ocurrencias de los patrones dados.

`+(`*lista-de-patrones*`)`
: Coincide con una o más ocurrencias de los patrones dados.

`@(`*lista-de-patrones*`)`
: Coincide con uno de los patrones dados.

`!(`*lista-de-patrones*`)`
: Coincide con cualquier cosa menos uno de los patrones dados.

Las coincidencias extendidas de patrón complicadas con cadenas largas es lenta, especialmente cuando los patrones contienen alteraciones y las cadenas contienen varias coincidencias. Usar coincidencias separadas sobre cadenas más cortas o usar vectores de cadenas en vez de una única cadena larga, puede ser más rápido.

3.5.9 Eliminación de Comillas

Después de las expansiones precedentes, todas las ocurrencias sin entrecomillar de los caracteres '\', '`'`' y '`"`' que no se produjeron a partir de una de las expresiones anteriores son eliminadas.

3.6 Redirecciones

Antes de que se ejecute una instrucción, su entrada y su salida pueden ser *redirigidas* usando una notación especial interpretada por el intérprete. La redirección permite a manejadores

de archivos de instrucciones ser duplicados, abiertos, cerrados, referidos a otros archivos, y puede cambiar los archivos de los que la instrucción lee y a los que escribe. La redirección también se puede usar para modificar los manejadores de archivos en el actual entorno de ejecución del intérprete. Los siguientes operadores de redirección pueden preceder o aparecer en cualquier lugar dentro de una instrucción simple o pueden seguir a una instrucción. Las redirecciones se procesan en el orden en que aparecen, de izquierda a derecha.

Cada redirección que pueda ser precedida de un número de archivo puede ser en su lugar precedida por una palabra con la forma {*nombre-de-var*}. En este caso, para cada operador de redirección a excepción de >&- y <&-, el intérprete reservará un descriptor de archivo mayor que 10 y lo asignará a {*nombre-de-var*}. Si >&- or <&- es precedido por {*nombre-de-var*}, el valor de *nombre-de-var* define el descriptor de archivo a cerrar. Si se proporciona {*nombre-de-var*}, la redirección persiste más allá del alcance de la instrucción, permitiendo al programador del intérprete gestionar el tiempo de vida del descriptor de archivo de forma manual.

En las siguientes descripciones, si se omite el número del descriptor de archivo y el primer carácter del operador de redirección es '<', la redirección hace referencia a la entrada estándar (descriptor de archivo 0). Si el primer carácter del operador de redirección es '>', la redirección hace referencia a la salida estándar (descriptor de archivo 1).

La palabra que sigue al operador de redirección en las siguientes descripciones, a no ser que se indique lo contrario, está sujeta a la expansión de llaves, expansión de virgulilla, expansión de parámetros, sustitución de instrucciones, expansión aritmética, eliminación de comillas, expansión de nombre de archivo y división de palabras. Si se expande a más de una palabra, Bash informa de un error.

Observe que el orden de las redirecciones es importante. Por ejemplo, la instrucción

```
ls > lista-de-dir 2>&1
```

dirige tanto la salida estándar (descriptor de archivo 1) como el error estándar (descriptor de archivo 2) al archivo *lista-de-dir*, mientras que la instrucción

```
ls 2>&1 > lista-de-dir
```

dirige solo la salida estándar al archivo *lista-de-dir*, porque el error estándar fue convertido en una copia de la salida estándar antes de que la salida estándar fuera redirigida a *lista-de-dir*.

Bash maneja varios nombres de archivos de forma especial cuando son usados en redirecciones, como se describe en la siguiente tabla. Si el sistema operativo en que Bash se ejecuta proporciona estos archivos especiales, Bash los usará; en caso contrario, los emulará internamente con el comportamiento descrito a continuación.

`/dev/fd/da`
: Si *da* es un entero válido, se duplica el descriptor de archivo *da*.

`/dev/stdin`
: Se duplica el descriptor de archivo 0.

`/dev/stdout`
: Se duplica el descriptor de archivo 1.

`/dev/stderr`
: Se duplica el descriptor de archivo 2.

`/dev/tcp/`*anfitrión*`/`*puerto*
: Si *anfitrión* es un nombre válido de anfitrión o dirección de Internet y *puerto* es un número entero de puerto o nombre de servicio, Bash trata de abrir el socket TCP correspondiente.

`/dev/udp/`*anfitrión*`/`*puerto*
: Si *anfitrión* es un nombre válido de anfitrión o dirección de Internet y *puerto* es un número entero de puerto o nombre de servicio, Bash trata de abrir el socket UDP correspondiente.

Un fallo al crear o abrir un archivo hace que la redirección falle.

Las redirecciones usando descriptores de archivo mayores que 9 deberían ser usadas con cuidado, ya que pueden entrar en conflicto con descriptores de archivo que el intérprete usa internamente.

3.6.1 Redirigiendo Entrada

La redirección de entrada hace que los archivos cuyo nombre deriva de la expansión de *palabra* sean abiertos para lectura en el descriptor de archivo `n`, o la entrada estándar (descriptor de archivo 0) si `n` no está especificado.

El formato generar para redirigir una entrada es:

`[`*n*`]<`*palabra*

3.6.2 Redirigiendo Salida

La redirección de salida hace que el archivo cuyo nombre deriva de la expansión de *palabra* sea abierto para escritura en el descriptor de archivo *n*, o el error estándar (descriptor de archivo 1) si no se especifica *n*. Si el archivo no existe, se crea; si existe, es truncado a tamaño cero.

El formato general para redirigir una salida es:

`[`*n*`]>[|]`*palabra*

Si el operador de redirección es '`>`' y la opción `noclobber` de la instrucción integrada `set` ha sido habilitada, la redirección fallará si el archivo cuyo nombre deriva de la expansión de *palabra* existe y es un archivo normal. Si el operador de redirección es '`>|`' o si el operador de redirección es '`>`' y no está habilitada la opción `noclobber`, la redirección se intenta incluso si el archivo nombrado por *palabra* existe.

3.6.3 Añadiendo Salida Redirigida

La redirección de salida de esta manera hace que el archivo cuyo nombre deriva de la expansión de *palabra* sea abierto para añadir en el descriptor de archivo *n*, o la salida estándar (descriptor de archivo 1), si no se especifica *n*. Si el archivo no existe, se crea.

El formato general para añadir una salida es:

`[`*n*`]>>`*palabra*

3.6.4 Redirigiendo Error Estándar y Salida Estándar

Esta construcción permite que sean redirigidas tanto la salida estándar (descriptor de archivo 1) como la salida de error estándar (descriptor de archivo 2) al archivo cuyo nombre es la expansión de *palabra*.

Hay dos formatos para redirigir error estándar y salida estándar:

```
&>palabra
```

y

```
>&palabra
```

De las dos formas, se prefiere la primera. Esto es equivalente semánticamente a

```
>palabra 2>&1
```

Al usar la segunda forma, puede que la *palabra* no se expanda a un número o a '`-`'. Si lo hace, se aplican otros operadores de redirección (véase Duplicando Descriptores de Archivo abajo) por razones de compatibilidad.

3.6.5 Añadiendo la Salida Estándar y el Error Estándar

Esta construcción permite que sean añadidas tanto la salida estándar (descriptor de archivo 1) como la salida de error estándar (descriptor de archivo 2) al archivo cuyo nombre es la expansión de *palabra*.

El formato para añadir la salida estándar y el error estándar es:

```
&>>palabra
```

Esto es equivalente semánticamente a

```
>>palabra 2>&1
```

(véase Duplicando Descriptores de Archivo abajo).

3.6.6 Documentos Aquí

Este tipo de redirección le ordena al intérprete leer la entrada de la fuente actual hasta que se encuentra una línea que contiene solo *palabra* (sin blancos al final). Todas las líneas leídas hasta ese punto son entonces usadas como la entrada estándar (o descriptor de archivo *n* si se especifica *n*) para una instrucción.

El formato del documento-aquí es:

```
[n]<<[−]palabra
        documento-aquí
delimitador
```

No se realiza expansión de parámetro y variable, sustitución de instrucción, expansión aritmética ni expansión de nombre de archivo en *palabra*. Si se entrecomilla alguna parte de *palabra* el *delimitador* es el resultado de la eliminación de comillas en *palabra*, y no se expanden las líneas en el documento-aquí. Si *palabra* no está entrecomillada, todas las líneas del documento aquí están sujetas a la expansión de parámetro, sustitución de instrucción y expansión aritmética, se ignora la secuencia de caracteres `\nueva línea` y '`/`' se debe usar para mencionar los caracteres '`\`', '`$`' y '`‘`'.

Si el operador de redirección es '`<<-`', todos los caracteres de tabulación iniciales son eliminados de las líneas de entrada y la línea que contiene *delimitador*. Esto permite indentar documentos-aquí en guiones de forma natural.

3.6.7 Cadenas Aquí

Una variante de documentos aquí, el formato es:

```
[n]<<< palabra
```

palabra experimenta la expresión de virgulilla, la expansión de parámetro y variable, la sustitución de instrucción, la expansión aritmética y la eliminación de comillas. No se realizan la expansión de nombre de archivo ni la separación de palabras. El resultado se proporciona a la instrucción como una sola cadena, con una nueva línea añadida, en su entrada estándar (o el descriptor de archivo *n* si se especifica *n*).

3.6.8 Duplicando Descriptores de Archivo

El operador de redirección

```
[n]<&palabra
```

se usa para duplicar descriptores de archivo de entrada. Si *palabra* se expande a uno o más dígitos, el descriptor de archivo denotado por *n* se vuelve una copia de ese descriptor de archivo. Si los dígitos en *palabra* no especifican un descriptor de archivo abierto para entrada, ocurre un error de redirección. Si *palabra* se evalúa a '`-`', se cierra el descriptor de archivo *n*. Si no se especifica *n*, se usa la entrada estándar (descriptor de archivo 0).

El operador

```
[n]>&palabra
```

se usa de forma parecida para duplicar la salida de descriptores de archivo. Si no se especifica *n*, se usa la salida estándar (descriptor de archivo 1). Si los dígitos en *palabra* no especifican un descriptor de archivo abierto para salida, ocurre un error de redirección. Si *palabra* se evalúa a '`-`', se cierra el descriptor de archivo *n*. Como caso especial, si se omite *n* y *palabra* no se expande a uno o más dígitos o '`-`', la salida estándar y el error estándar son redirigidos como se describió anteriormente.

3.6.9 Moviendo Descriptores de Archivo

El operador de redirección

```
[n]<&dígito-
```

mueve el descriptor de archivo *dígito* al descriptor de archivo *n*, o a la entrada estándar (descriptor de archivo 0) si no se especifica *n*. *dígito* se cierra tras ser duplicado a *n*.

De forma similar, el operador de redirección

```
[n]>&dígito-
```

mueve el descriptor de archivo *dígito* al descriptor de archivo *n*, o a la salida estándar (descriptor de archivo 1) si no se especifica *n*.

3.6.10 Abriendo Descriptores de Archivo para Leer y Escribir

El operador de redirección

```
[n]<>palabra
```

hace que el archivo cuyo nombre es la expansión de *palabra* sea abierto tanto para leer como para escribir en el descriptor de archivo *n* o en el descriptor de archivo 0 si no se especifica *n*. Si el archivo no existe, se crea.

3.7 Ejecutando Instrucciones

3.7.1 Expansión de Instrucciones Simples

Cuando se ejecuta una instrucción simple, el intérprete realiza las siguientes expansiones, asignaciones y redirecciones, de izquierda a derecha, en el siguiente orden.

1. Las palabras que el analizador ha marcado como asignaciones de variables (aquellas que preceden al nombre de la instrucción) y redirecciones se guardan para su tratamiento posterior.
2. Las palabras que no son asignaciones de variable o redirecciones son expandidas (véase Sección 3.5 [Expansiones del Intérprete], página 24). Si alguna palabra permanece tras la expansión, la primera palabra se toma como el nombre de la instrucción y las demás palabras son los argumentos.
3. Las redirecciones se realizan como se describió anteriormente (véase Sección 3.6 [Redirecciones], página 37).
4. El texto después del '`=`' en cada asignación de variable experimenta la expansión de virgulilla, expansión de parámetro, sustitución de instrucciones, expansión aritmética y eliminación de comillas antes de ser asignado a una variable.

Si no se produce un nombre de instrucción, las asignaciones de variables afectan al entorno actual del intérprete. De lo contrario, las variables son añadidas al entorno de la instrucción ejecutada y no afectan al entorno del intérprete actual. Si cualquiera de las asignaciones intenta asignar un valor a una variable de solo lectura, ocurre un error y la instrucción sale con un estado de error distinto a cero.

Si no se produce un nombre de instrucción, se realizan redirecciones, pero no afectan al entorno actual del intérprete. Un error de redirección hace que la instrucción se cierre con un estado distinto de cero.

Si hay un nombre de instrucción sobrante después de la expansión, la ejecución procede como se describe más adelante. De lo contrario, la instrucción finaliza. Si una de las expansiones contenía una sustitución de instrucción, el estado de salida de la instrucción es el estado de salida de la última sustitución de instrucción realizada. Si no había ninguna sustitución, la instrucción finaliza con un estado de cero.

3.7.2 Búsqueda y Ejecución de Instrucciones

Después de que una instrucción haya sido dividida en palabras, si resulta en una instrucción simple y una lista opcional de argumentos, se toman las siguientes acciones.

1. Si el nombre de la instrucción no contiene barras, el intérprete trata de localizarla. Si existe una función del intérprete con ese nombre, esa función se llama como se describe en Sección 3.3 [Funciones del Intérprete], página 19.
2. Si el nombre no coincide con una función, el intérprete lo busca en la lista de instrucciones integradas del intérprete. Si se encuentra una coincidencia, se llama a esa instrucción integrada.
3. Si el nombre no es ni una función del intérprete ni una instrucción integrada y no contiene barras, Bash busca un directorio que contenga un archivo ejecutable con ese nombre en cada elemento de `$PATH`. Bash usa una tabla hash para recordar los nombres de ruta completos de archivos ejecutables para evitar múltiples búsquedas `PATH`

(vea la descripción de `hash` en Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48). Solo se realiza una búsqueda de los directorios en `$PATH` si la instrucción no se encontró en la tabla hash. Si la búsqueda es infructuosa, el intérprete busca una función del intérprete definida llamada `command_not_found_handle`. Si esa función existe, se llama en un entorno de ejecución separado con la instrucción original y los argumentos de la instrucción original como sus argumentos, y el estado de salida de la función se convierte en el estado de salida de ese subintérprete. Si no está definida esa función, el intérprete imprime un mensaje de error y devuelve un estado de salida de 127.

4. Si la búsqueda es exitosa o si el nombre de la instrucción contiene una o más barras, el intérprete ejecuta el programa nombrado en un entorno de ejecución aislado. El argumento 0 se establece al nombre dado, y el resto de argumentos para la instrucción se establecen a los argumentos proporcionados, si hay.
5. Si esta ejecución falló porque el archivo no está en formato ejecutable, y el archivo no es un directorio, se toma por un *guion de intérprete* y el intérprete lo ejecuta como se describe en Sección 3.8 [Guiones del Intérprete], página 46.
6. Si la instrucción no fue iniciada asíncronamente, el intérprete espera a que la instrucción se complete y recoge su estado de salida.

3.7.3 Entorno de Ejecución de Instrucciones

El intérprete tiene un *entorno de ejecución*, que consta de lo siguiente:

- abre archivos heredados por el intérprete durante la llamada, como se modificaron por redirecciones proporcionadas a la instrucción integrada `exec`
- el directorio de trabajo actual como fue establecido por `cd`, `pushd` o `popd`, o heredado por el intérprete durante la llamada
- el modo de máscara de creación de archivo como fue establecido por `umask` o heredado del padre del intérprete
- traps actuales establecidas por `trap`
- parámetros del intérprete que son establecidos mediante asignación de variables o con `set` o heredados del padre del intérprete en el entorno
- funciones del intérprete definidas durante la ejecución o heredadas del padre del intérprete en el entorno
- opciones habilitadas en la llamada (o predeterminadas o con argumentos de línea de órdenes) o mediante `set`
- opciones habilitadas por `shopt` (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73)
- aliases del intérprete definidos con `alias` (véase Sección 6.6 [Aliases], página 105)
- varios IDs de procesos, incluyendo aquellos de las tareas en segundo plano (véase Sección 3.2.4 [Listas], página 10), el valor de `$$` y el valor de `$PPID`

Cuando una instrucción simple distinta de una instrucción integrada o una función del intérprete va a ser ejecutada, es llamada en un entorno de ejecución aislado que consta de lo siguiente. A no ser que se indique lo contrario, los valores se heredan del intérprete.

- los archivos abiertos del intérprete, más las modificaciones y adiciones especificadas por redirecciones a la instrucción

- el directorio de trabajo actual
- el modo de la máscara de creación de archivos
- las variables y funciones del intérprete marcadas para exportar, junto a variables exportadas para la instrucción, pasadas en el entorno (véase Sección 3.7.4 [Entorno], página 44)
- las traps atrapadas por el intérprete son restablecidas a los valores heredados del padre del intérprete, y las traps ignoradas por el intérprete son ignoradas

Una instrucción llamada en este entorno aislado no puede afectar al entorno de ejecución del intérprete.

La sustitución de instrucciones, las instrucciones agrupadas con paréntesis y las instrucciones asíncronas son llamadas en un entorno de subintérprete que es una copia del entorno del intérprete, a excepción de que las traps atrapadas por el intérprete son restablecidas a los valores que el intérprete heredó de su padre durante la llamada. Las instrucciones integradas que son llamadas como parte de una tubería también son ejecutadas en un entorno de subintérprete. Los cambios hechos al entorno del subintérprete no pueden afectar al entorno de ejecución del intérprete.

Los subintérpretes generados para ejecutar sustituciones de instrucciones heredan el valor de la opción `-e` del padre del intérprete. Cuando no está en modo POSIX, Bash borra la opción `-e` en dichos subintérpretes.

Si una instrucción es seguida por un '`&`' y no está activado el control de tareas, la entrada estándar para el instrucción es el archivo vacío `/dev/null`. De lo contrario, la instrucción llamada hereda los descriptores de archivo del intérprete ejecutor según se modifican mediante redirección.

3.7.4 Entorno

Cuando se llama un programa se le pasa un vector de cadenas llamado el *entorno*. Esto es una lista de pares nombre-valor, de la forma `nombre=valor`.

Bash proporciona varias formas de manipular el entorno. Durante la llamada, el intérprete escanea su propio entorno y crea un parámetro para cada nombre encontrado, automáticamente marcándolo para *exportar* a los procesos hijos. Las instrucciones ejecutadas heredan el entorno. Las instrucciones `export` y '`declare -x`' permiten añadir y eliminar parámetros y funciones en el intérprete. Si se modifica el valor de un parámetro en el intérprete, el nuevo valor se vuelve parte del entorno, reemplazando al viejo. El entorno heredado por cualquier instrucción ejecutada consta del entorno inicial del intérprete, cuyos valores pueden ser modificados en el intérprete, menos las parejas eliminadas mediante las instrucciones `unset` y '`export -n`', más las adiciones mediante las instrucciones `export` y '`declare -x`'.

El entorno para cualquier instrucción simple o función puede ser aumentado temporalmente prefijándolo con asignaciones de parámetros, como se describe en Sección 3.4 [Parámetros del Intérprete], página 21. Estas sentencias de asignación afectan solo al entorno visto por esa instrucción.

Si está habilitada la opción `-k` (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68), todas las asignaciones de parámetros son ubicadas en el entorno para una instrucción, no solo aquellas que preceden al nombre de la instrucción.

Cuando Bash llama a una instrucción externa, la variable '`$_`' es establecida al nombre de ruta completo de la instrucción y pasado a esa instrucción en su entorno.

3.7.5 Estado de Salida

El estado de salida de una instrucción ejecutada es el valor devuelto por la llamada de sistema *waitpid* o función equivalente. Los estados de salida se sitúan entre 0 y 255, sin embargo, como se explica a continuación, el intérprete puede usar los valores encima de 125 de forma especial. Los estados de salida de instrucciones integradas del intérprete y instrucciones compuestas también están limitadas a este rango. Bajo ciertas condiciones, el intérprete usará valores especiales para indicar modos de fallo específicos.

Para propósitos del intérprete, una instrucción que salga con un estado de salida de cero ha tenido éxito. Un estado de salida distinto de cero indica fallo. Este esquema aparentemente en contra del sentido común se usa para que haya un modo bien definido de indicar éxito y una variedad de formas de indicar varios modos de fallo. Cuando una instrucción termina con una señal fatal cuyo número es N, Bash usa el valor 128+N como el estado de salida.

Si no se encuentra una instrucción, el proceso hijo creado para ejecutarla devuelve un estado de 127. Si se encuentra una instrucción pero no es ejecutable, el estado de retorno es 126.

Si una instrucción falla a causa de un error durante la expansión o redirección, el estado de salida es mayor que cero.

El estado de salida es usado por las instrucciones condicionales de Bash (véase Sección 3.2.5.2 [Construcciones Condicionales], página 12) y algunas de las construcciones de listas (véase Sección 3.2.4 [Listas], página 10).

Todas las instrucciones integradas de Bash devuelven un estado de salida de cero si tienen éxito y un estado de salida distinto de cero en caso de fallo, de forma que pueden ser usados por las construcciones condicionales y de listas. Todas las instrucciones integradas devuelven un estado de salida de 2 para indicar un uso incorrecto, generalmente opciones inválidas o argumentos que faltan.

3.7.6 Señales

Cuando Bash es interactivo, en ausencia de traps, ignora `SIGTERM` (de forma que '`kill 0`' no mate un intérprete interactivo) y `SIGINT` se atrapa y se maneja (de forma que la instrucción integrada `wait` no se pueda interrumpir). Cuando Bash recive una `SIGINT`, sale de cualquier bucle que se ejecute. En todos los casos, Bash ignora `SIGQUIT`. Si está en efecto el control de tareas (véase Capítulo 7 [Control de Tareas], página 118), Bash ignora `SIGTTIN`, `SIGTTOU` y `SIGTSTP`.

Las instrucciones que no son integradas iniciadas por Bash tienen manejadores de señales establecidos a los valores heredados por el intérprete de su padre. Cuando no está en efecto el control de tareas, las instrucciones asíncronas ignoran `SIGINT` y `SIGQUIT` además de estos manejadores heredados. Las instrucciones ejecutadas como un resultado de la sustitución de instrucciones ignoran las señales de control de tareas generadas con el teclado `SIGTTIN`, `SIGTTOU` y `SIGTSTP`.

El intérprete se cierra por defecto al recibir un `SIGHUP`. Antes de cerrarse, un intérprete interactivo reenvía la `SIGHUB` a todas las tareas, en ejecución o detenidas. A las tareas

detenidas se les envía `SIGCONT` para asegurarse de que reciben la `SIGHUB`. Para evitar que el intérprete envíe la señal `SIGHUB` a una tarea concreta, debería ser eliminada de la tabla de tareas con la instrucción integrada `disown` (véase Sección 7.2 [Instrucciones Integradas de Control de Tareas], página 119) o marcada para no recibir `SIGHUB` usando `disown -h`.

Si se ha habilitado la opción `huponexit` con `shopt` (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73), Bash manda una `SIGHUB` a todas las tareas cuando se cierra un intérprete de acceso interactivo.

Si Bash está esperando a que una instrucción se complete y recibe una señal para la cual una trap ha sido establecida, el trap no será ejecutada hasta que la instrucción se complete. Cuando Bash está esperando a una instrucción asíncrona mediante la instrucción integrada `wait`, la recepción de una señal para la cual una trap ha sido establecida hará que la instrucción integrada `wait` retorne inmediatamente con un estado de salida mayor que 128, inmediatamente después se ejecuta la trap.

3.8 Guiones del Intérprete

Un guion del intérprete es un archivo de texto que contiene instrucciones del intérprete. Cuando se usa tal archivo como el primer argumento no opcional al llamar a Bash, y ni la opción `-c` ni la `-s` es proporcionada (véase Sección 6.1 [Llamando a Bash], página 95), Bash lee y ejecuta instrucciones del archivo, después se cierra. Este modo de operación crea un intérprete no interactivo. El intérprete busca primero el archivo en el directorio actual, y busca en los directorios en `PATH` si no es encontrado ahí.

Cuando Bash ejecuta un guion del intérprete, establece el parámetro especial `0` al nombre del archivo, en vez de al nombre del intérprete, y los argumentos restantes se asignan a los parámetros posicionales, si se pasa alguno. Si no se proporcionan argumentos adicionales, se eliminan los parámetros posicionales.

Un guion del intérprete puede hacerse ejecutable usando la instrucción `chmod` para activar el bit de ejecución. Cuando Bash encuentra tal archivo al buscar una instrucción en el `$PATH`, genera un subintérprete para ejecutarlo. En otras palabras, ejecutar

```
nombre-de-archivo argumentos
```

es equivalente a ejecutar

```
bash nombre-de-archivo argumentos
```

si `nombre-de-archivo` es un guion del intérprete ejecutable. Este subintérprete se reinicializa, para que el efecto sea como si un nuevo intérprete hubiera sido llamado para interpretar el guion, con la excepción de que las ubicaciones de las instrucciones recordadas por el padre (lea la descripción de `hash` en Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48) son retenidas por el hijo.

La mayoría de las versiones de Unix hacen que esto sea una parte del mecanismo de ejecución de instrucciones del sistema. Si la primera línea del guion comienza con los dos caracteres '`#!`', el resto de la línea especifica un intérprete para el programa y, dependiendo del sistema operativo, uno o más argumentos opcionales para ese intérprete. Así, puedes especificar Bash, `awk`, Perl o cualquier otro intérprete y escribir el resto del archivo del guion en ese lenguaje.

Los argumentos para el intérprete consisten en uno o más argumentos opcionales que siguen al nombre del intérprete en la primera línea de un archivo de guion, seguido por el

nombre del archivo de guion, seguido del resto de argumentos proporcionados al guion. Los detalles de cómo se divide la línea del intérprete entre el nombre del intérprete y un conjunto de argumentos varían entre sistemas. Bash realizará esta acción en sistemas operativos que por sí mismos no la realizan. Téngase en cuenta que las versiones antiguas de Unix limitan el nombre del intérprete y un argumento independiente a un máximo de 32 caracteres, así que no es posible asumir que usar más de un argumento funcionará.

Los guiones de Bash normalmente comienzan por `#! /bin/bash` (asumiendo que Bash haya sido instalado en `/bin`), puesto que esto asegura que Bash será usado para interpretar el guion, incluso si se ejecuta por otro intérprete. Es una expresión común el uso de `env` para encontrar `bash` incluso si ha sido instalado en otro directorio: `#!/usr/bin/env bash` encontrará la primera ocurrencia de `bash` en `$PATH`.

4 Instrucciones Integradas del Intérprete

Las instrucciones integradas están incluidas en el propio intérprete. Cuando el nombre de una instrucción integrada se usa como la primera palabra de una instrucción simple (véase Sección 3.2.2 [Instrucciones Simples], página 8), el intérprete ejecuta la orden directamente, sin invocar otro programa. Las instrucciones integradas son necesarias para implementar funcionalidad imposible o inconveniente de obtener con utilidades separadas.

Esta sección describe brevemente las instrucciones integradas que Bash hereda del Bourne Shell, así como las instrucciones integradas que son únicas o han sido extendidas en Bash.

En otros capítulos se describen varias instrucciones integradas: las instrucciones integradas que proporcionan la interfaz de Bash para las utilidades del control de tareas (véase Sección 7.2 [Instrucciones Integradas de Control de Tareas], página 119), la pila de directorio (véase Sección 6.8.1 [Instrucciones Integradas de la Pila de Directorios], página 108), el historial de instrucciones (véase Sección 9.2 [Instrucciones Integradas del Historial de Bash], página 162) y las utilidades de completado programables (véase Sección 8.7 [Instrucciones Integradas de Compleción Programable], página 153).

Muchas de las instrucciones integradas han sido extendidas por POSIX o Bash.

A no ser que se indique lo contrario, cada instrucción integrada documentada como que acepta opciones precedidas por '`-`' acepta '`--`' para indicar el fin de las opciones. Las instrucciones integradas `:`, `true`, `false` y `test` / `[` no aceptan opciones y no tratan '`--`' de forma especial. Las instrucciones integradas `exit`, `logout`, `return`, `break`, `continue`, `let` y `shift` aceptan y procesan argumentos que empiezan con '`-`' sin requerir '`--`'. Otras instrucciones integradas que aceptan argumentos pero no especifican que aceptan opciones, interpretan argumentos que empiezan con '`-`' como opciones inválidas y requieren '`--`' para evitar esta interpretación.

4.1 Instrucciones Integradas del Bourne Shell

Las siguientes instrucciones integradas del intérprete son heredadas del Bourne Shell. Estas órdenes se implementan como está especificado en el estándar POSIX.

`:` (dos puntos)

```
: [argumentos]
```

No hace nada más allá de expandir *argumentos* y realizar redirecciones. El estado de retorno es cero.

`.` (un punto)

```
. nombre-de-archivo [argumentos]
```

Lee y ejecuta instrucciones del argumento *nombre-de-archivo* en el actual contexto del intérprete. Si *nombre-de-archivo* no contiene una barra, la variable `PATH` se usa para encontrar *nombre-de-archivo*. Cuando Bash no está en modo POSIX, se explora el directorio actual si *nombrearchivo* no se encuentra en `PATH`. Si se pasan *argumentos*, se convierten en los argumentos posicionales cuando se ejecuta *nombre-de-archivo*. De modo contrario, los argumentos posicionales no se cambian. Si la opción `-T` está habilitada, `source` hereda cualquier trap en `DEBUG`; si no, cualquier cadena de `DEBUG` trap se guarda y se restaura alrededor

de la llamada a `source`, y `source` elimina la trap `DEBUG` mientras se ejecuta. Si `-T` no está establecido, y el archivo leído cambia el trap `DEBUG`, el nuevo valor se almacena cuando `source` termina. El estado de retorno es el estado de salida de la última instrucción ejecutada, o cero si no se ejecuta ninguna orden. Si no se encuentra *nombre-de-archivo*, o no se puede leer, el estado de retorno es diferente de cero. Esta instrucción integrada es equivalente a `source`.

`break`

```
break [n]
```

Sale de un bucle `for`, `while`, `until` o `select`. Si se proporciona *n*, se sale del nivel *n* de anidación del bucle. *n* debe ser mayor o igual a 1. El estado de salida es cero a no ser que *n* no sea mayor o igual a 1.

`cd`

```
cd [-L|[-P [-e]] [-@] [directorio]
```

Cambia el actual directorio de trabajo a *directorio*. Si no se proporciona *directorio*, se usa el valor de la variable del intérprete `HOME`. Cualquier argumento adicional que siga a *directorio* es ignorado. Si la variable de intérprete `CDPATH` existe, es usada como ruta de búsqueda: en cada nombre de directorio de `CDPATH` se busca *directorio*, con los nombres alternativos de directorios en `CDPATH` separados por dos puntos (‘`:`’). Si *directorio* empieza por una barra, `CDPATH` no se usa.

La opción `-P` significa que no sigue los enlaces simbólicos: los enlaces simbólicos son resueltos mientras `cd` está recorriendo *directorio* y antes de procesar una instancia de ‘`..`’ en *directorio*.

Por defecto, o cuando se proporciona la opción `-L`, los enlaces simbólicos en *directorio* son resueltos después de que `cd` procesa una instancia de ‘`..`’ en *directorio*.

Si ‘`..`’ aparece en *directorio*, se procesa eliminando el componente de nombre de ruta que le precede inmediatamente, de vuelta a una barra o al inicio de *directorio*.

Si la opción `-e` es proporcionada junto a `-P` y el directorio de trabajo actual no se puede determinar con éxito después de un exitoso cambio de directorio, `cd` revolverá un estado de retorno no exitoso.

En sistemas que lo soportan, la opción `-@` presenta los atributos extendidos asociados a un archivo como directorio.

Si *directorio* es ‘`-`’, este se convierte en `$OLDPWD` antes de que se intente el cambio de directorio.

Si se usa un nombre de directorio de `CDPATH` no vacío o si ‘`-`’ es el primer argumento y el cambio de directorio es exitoso, el nombre de ruta absoluto del nuevo directorio de trabajo se escribe a la salida estándar.

El código de retorno es cero si el directorio es cambiado exitosamente, distinto de cero de no ser así.

`continue`

```
continue [n]
```

Reanuda la siguiente iteración de un bucle `for`, `while`, `until` o `select` que lo contiene. Si se proporciona *n*, reanuda la ejecución del bucle de nivel de anidamiento *n* que lo contiene. *n* debe ser mayor o igual a 1. El código de retorno es cero a no ser que *n* no sea mayor o igual a 1.

`eval`

`eval [`*arguments*`]`

Los argumentos son concatenados juntos en una única instrucción, que entonces es leída y ejecutada, y su estado de salida devuelto como el estado de salida de `eval`. Si no hay argumentos o solo argumentos vacíos, el estado de retorno es cero.

`exec`

`exec [-cl] [-a` *nombre*`] [`*instrucción* `[`*argumentos*`]]`

Si se proporciona *instrucció*, reemplaza el intérprete sin crear un nuevo proceso. Si se proporciona la opción `-l`, el intérprete pone una barra al comienzo del argumento número cero pasado a *instrucción*. Esto es lo que hace el programa `login`. La opción `-c` hace que *instrucción* sea ejecutada con un entorno vacío. Si se proporciona `-a`, el intérprete pasa *nombre* como el argumento número cero a *instrucción*. Si por alguna razón no se puede ejecutar *instrucción*, se cierra un intérprete no interactivo, a no ser que esté habilitada la opción del intérprete `execfail`. En ese caso devuelve fallo. Un intérprete interactivo devuelve fallo si el archivo no puede ser ejecutado. Un subintérprete finaliza de forma incondicional si `exec` falla. Si no se especifica ninguna *instrucción*, se pueden usar redirecciones para afectar al entorno actual del intérprete. Si no hay errores de redirección, el estado de retorno es cero; de lo contrario, el estado de retorno es distinto a cero.

`exit`

`exit [`*n*`]`

Sale del intérprete, devolviendo un estado de *n* al padre del intérprete. Si se omite *n*, el estado de salida es el mismo de la última orden ejecutada. Cualquier trap en `EXIT` se ejecuta antes de que el intérprete termine.

`export`

`export [-fn] [-p] [`*nombre*`[=`*valor*`]]`

Marca cada *nombre* para que sea pasado a los procesos hijos en el entorno. Si se proporciona la opción `-f`, los *nombres* se refieren a funciones del intérprete; de lo contrario, los nombres se refieren a variables del intérprete. La opción `-n` equivale a dejar de marcar cada *nombre* para exportación. Si no se proporcionan *nombres*, o si se pasa la opción `-p`, se muestra una lista de nombres de todas las variables exportadas. La opción `-p` muestra la salida en una forma que puede ser reusada como entrada. Si a un nombre de variable sigue =*valor*, el valor de la variable se establece a *valor*.

El estado de retorno es cero a no ser que se proporcione una opción inválida, uno de los nombres no sea un nombre de variable de intérprete válido o se proporcione `-f` con un nombre que no sea una función del intérprete.

`getopts`

```
getopts cadenaopc nombre [arg ...]
```

`getopts` se usa por los guiones del intérprete para analizar parámetros posicionales. *cadenaopc* contiene los caracteres de opción que serán reconocidos; si a un carácter le siguen dos puntos, se espera que la opción tenga un argumento, que debería estar separado de ese por un espacio. Los dos puntos (‘`:`’) y el símbolo de interrogación (‘`?`’) no pueden ser usados como caracteres de opción. Cada vez que se invoca, `getopts` ubica la siguiente opción en la variable de intérprete *nombre*, inicializando *nombre* si no existe, y el índice del siguiente argumento a ser procesado en la variable `OPTIND`. `OPTIND` es inicializado a 1 cada vez que el intérprete o un guion de intérprete es invocado. Cuando una opción requiere un parámetro, `getopts` ubica ese argumento en la variable `OPTARG`. El intérprete no elimina `OPTIND` automáticamente; tiene que ser eliminado manualmente entre múltiples llamadas a `getopts` dentro de la misma invocación del intérprete si se pretende usar un nuevo conjunto de parámetros.

Cuando se encuentra el fin de las opciones, `getopts` sale con un valor de retorno mayor que cero. Se establece `OPTIND` al índice del primer argumento que no sea una opción y *nombre* se establece a ‘`?`’.

`getopts` normalmente procesa los parámetros posicionales, pero si se proporcionan más argumentos como valores *arg*, `getopts` procesa esos en su lugar.

`getopts` puede informar de errores de dos formas. Si el primer carácter de *cadenaopc* es dos puntos, se usa el aviso de errores *silencioso*. En funcionamiento normal, los mensajes de diagnóstico se imprimen cuando se encuentran opciones inválidas o argumentos de opción que faltan. Si se establece la variable `OPTERR` a 0, no se mostrarán mensajes de error, incluso si el primer carácter de `cadenaopc` no es dos puntos.

Si se detecta una opción inválida, `getopts` guarda ‘`?`’ en *nombre* y, si no es silencioso, imprime un mensaje de error y elimina `OPTARG`. Si `getopts` es silencioso, el carácter de opción encontrado se guarda en `OPTARG` y no se muestra ningún mensaje de diagnóstico.

Si no se encuentra un argumento requerido, y `getopts` no es silencioso, se guarda un símbolo de interrogación (‘`?`’) en *nombre*, `OPTARG` se elimina, y se imprime un mensaje de diagnóstico. Si `getopts` es silencioso, se guardan dos puntos (‘`:`’) en *nombre* y `OPTARG` se establece al carácter de opción encontrado.

`hash`

```
hash [-r] [-p nombre-de-archivo] [-dt] [nombre]
```

Cada vez que se invoca `hash`, recuerda los nombres de ruta completos de las órdenes especificadas como argumentos *nombre*, de forma que no deben ser buscados en las próximas invocaciones. Las instrucciones se encuentran buscando a través de los directorios listados en `$PATH`. Cualquier nombre de ruta recordado previamente es descartado. La opción `-p` inhibe la búsqueda por ruta, y *nombre-de-archivo* se usa como la ubicación de *nombre*. La opción `-r` hace que el intérprete se olvide de todas las ubicaciones recordadas. La opción `-d` hace que el intérprete se olvide de la ubicación recordada de cada *nombre*. Si se

proporciona la opción `-t`, se imprime el nombre de ruta completo al que corresponde cada *nombre*. Si se proporcionan múltiples argumentos *nombre* con `-t`, el *nombre* se imprime antes del nombre ruta completo hecho hash. La opción `-l` hace que la salida se muestre en un formato que puede ser reutilizado como entrada. Si no se pasan argumentos, o si solo se proporciona `-l`, se imprime información sobre instrucciones recordadas. El estado de retorno es cero a no ser que *nombre* no se encuentre o se proporcione una opción inválida.

`pwd`

```
pwd [-LP]
```

Imprime el nombre de ruta absoluto del actual directorio de trabajo. Si se proporciona la opción `-P`, el nombre de ruta imprimido no contendrá enlaces simbólicos. Si se proporciona la opción `-L`, el nombre de ruta impreso puede contener enlaces simbólicos. El estado de retorno es cero a no ser que se produzca un error mientras se determina el nombre del directorio actual o se proporcione una opción inválida.

`readonly`

```
readonly [-aAf] [-p] [nombre[=valor]] ...
```

Marca cada *nombre* como de solo lectura. Los valores de estos nombres no pueden ser cambiados en asignaciones posteriores. Si se proporciona la opción `-f`, cada *nombre* se refiere a una función del intérprete. La opción `-a` significa que cada *nombre* se refiere a una variable del tipo vector indexado; la opción `-A` significa que cada *nombre* se refiere a una variable del tipo vector asociativo. Si se proporcionan ambas opciones, `-A` tiene precedencia. Si no se pasan argumentos *nombre*, o si se proporciona la opción `-p`, se imprime una lista de todos los nombres de solo lectura. Las otras opciones pueden ser usadas para restringir la salida a un subconjunto de los nombres de solo lectura. La opción `-p` hace que la salida sea mostrada en un formato que puede ser reutilizado como entrada. Si nombre de variable está seguido de =*valor*, el valor de la variable se establece a *valor*. El estado de retorno es cero a no ser que se proporcione una opción inválida, uno de los argumentos *nombre* no sea un nombre válido de variable de intérprete o de función o la opción `-f` se proporcione con un nombre que no sea una función del intérprete.

`return`

```
return [n]
```

Hace que una función del intérprete se deje de ejecutar y devuelva el valor *n* a su ejecutor. Si no se proporciona *n*, el valor de retorno es el estado de salida de la última instrucción ejecutada en la función. Si `return` es ejecutado por un manejador trap, la última instrucción usada para determinar el estado es la última instrucción ejecutada antes del manejador trap. Si se ejecuta `return` durante una trap del tipo `DEBUG`, la última instruccion usada para determinar el estado es la última instrucción ejecutada por el manejador trap antes de que `return` fuera invocado. `return` puede ser usado también para terminar la ejecución de un guion que está siendo ejecutado con la función integrada `.` (`source`), devolviendo o *n* o el estado de salida de la última instrucción

ejecutada dentro del guion como el estado de salida del guion. Si se proporciona *n*, el valor de retorno es sus 8 bits menos significativos. Cualquier instrucción asociada a la trap `RETURN` es ejecutada antes de que se reanude la ejecución tras la función o el guion. El estado de retorno es distinto a cero si `return` es proporcionado como un argumento no número o es usado fuera de una función y no durante la ejecución de un guion por `.` o `source`.

`shift`

```
shift [n]
```

Mueve los parámetros posicionales *n* a la izquierda. Los parámetros posicionales de *n*+1 ... `$#` se renombran a `$1` ... `$#`-*n*. Los parámetros representados por números de `$#` a `$#`-*n*+1 son deshabilitados. *n* debe ser un número no negativo menor o igual a `$#`. Si *n* es cero o mayor que `$#`, no se cambian los parámetros posicionales. Si no se proporciona *n*, se asume que es 1. El estado de retorno es cero a no ser que *n* sea mayor que `$#` o menor que cero, distinto a cero de lo contrario.

`test`
`[`

```
test expr
```

Evalúa una expresión condicional *expr* y devuelve un estado de 0 (verdadero) o 1 (falso). Cada operador y operando debe ser un argumento separado. Las expresiones están compuestas de las opciones primarias descritas más abajo en Sección 6.4 [Expresiones Condicionales de Bash], página 101. `test` no acepta ninguna opción, tampoco acepta e ignora un argumento de `--` como el significado del fin de las opciones.

Cuando se usa la forma `[`, el último argumento de la instrucción debe ser un `]`.

Las expresiones pueden ser combinadas usando los siguientes operadores, listados en orden descendente de precedencia. La evaluación depende del número de argumentos; consulta más abajo. La precedencia de operador se usa cuando hay cinco o más argumentos.

`! expr` Verdadero si *expr* es falso.

`( expr )` Devuelve el valor de *expr*. Esto puede ser usado para sobreescribir la precedencia normal de operadores.

expr1 `-a` *expr2*
: Verdadero si tanto *expr1* y *expr2* son verdadero.

expr1 `-o` *expr2*
: Verdadero si *expr1* o *expr2* es verdadero.

Las instrucciones internas `test` y `[` evalúan expresiones condicionales usando un conjunto de reglas basado en el número de argumentos.

0 argumentos
: La expresión es falsa.

1 argumento
: La expresión es verdadera solo si el argumento no es nulo.

2 argumentos
: Si el primer argumento es '`!`', la expresión es verdadera solamente si el segundo argumento es nulo. Si el primer argumento es uno de los operadores condicionales unarios (véase Sección 6.4 [Expresiones Condicionales de Bash], página 101), la expresión es verdadera si la comprobación unaria es verdadera. Si el primer argumento no es un operador unario válido, la expresión es falsa.

3 argumentos
: Las siguientes condiciones se aplican en el orden listado.

  1. Si el segundo argumento es alguno de los operadores binarios condicionales (véase Sección 6.4 [Expresiones Condicionales de Bash], página 101), el resultado de la expresión es el resultado de la comprobación binaria que usa el primer argumento y el tercero como operadores. Los operadores '`-a`' y '`-o`' son considerados operadores binarios cuando hay tres argumentos.
  2. Si el primer argumento es '`!`', el valor es la negación de la comprobación de dos argumentos usando el segundo argumento y el tercero.
  3. Si el primer argumento es exactamente '`(`' y el tercer argumento es exactamente '`)`', el resultado es la comprobación de un argumento del segundo argumento.
  4. De lo contrario, la expresión es falsa.

4 argumentos
: Si el primer argumento es '`!`', el resultado es la negación de la expresión de tres argumentos compuesta de los argumentos restantes. De lo contrario, la expresión es analizada y evaluada según la precedencia usando las reglas listadas anteriormente.

5 o más argumentos
: La expresión es analizada y evaluada según la precedencia usando las reglas listadas anteriormente.

Cuando se usa con `test` o '`[`', los operadores '`<`' y '`>`' ordenan lexicográficamente usando el orden ASCII.

`times`

```
times
```

Imprime los tiempos de usuario y de sistema usados por el intérprete y sus hijos. El estado de retorno es cero.

`trap`

```
trap [-lp] [arg] [id_señal ...]
```

Las instrucciones en *arg* están pensadas para ser leídas y ejecutadas cuando el intérprete recibe la señal *id_señal*. Si falta *parám* (y hay una señal *id_señal*) o es igual a '`-`', cada disposición de señal especificada es restablecida al valor que tenía cuando el intérprete fue iniciado. Si *arg* es la cadena nula, se ignora la señal

especificada por cada *id_señal* y las instrucciones que ejecuta. Si *arg* no está presente y `-p` ha sido proporcionado, el intérprete muestra las instrucciones trap asociadas con cada *id_señal*. Si no se proporcionan argumentos, o solo se proporciona `-p`, `trap` imprime la lista de instrucciones asociadas con cada número de señal en una forma que puede ser reusada como entrada para el intérprete. La opción `-l` hace que el intérprete imprima una lista de nombres de señales y sus números correspondientes. Cada *id_señal* es o bien un nombre de señal o un número de señal. Los nombres de señales no distinguen entre mayúsculas y minúsculas y el prefijo `SIG` es opcional.

Si una *id_señal* es `0` o `EXIT`, *arg* se ejecuta cuando el intérprete se cierra. Si un *id_señal* es `DEBUG`, se ejecuta la instrucción *arg* antes de cada instrucción simple, instrucción `for`, instrucción `case`, instrucción `select`, cada instrucción `for` aritmética y antes de que se ejecute la primera instrucción en una función del intérprete. Consulte la descripción de la opción `extdebug` de la instrucción integrada `shopt` (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73) para encontrar detalles de su efecto sobre la trap `DEBUG`. Si un *id_señal* es `RETURN`, la instrucción *arg* se ejecuta cada vez que una instrucción del intérprete o un guion ejecutado con las instrucciones integradas `.` o `source` terminan de ejecutarse.

Si el *sigspec* es `ERR`, la instrucción *arg* se ejecuta cada vez que una tubería (que puede consistir de una única instrucción simple), una lista o una instrucción compuesta devuelve un estado diferente de cero, de acuerdo a las siguientes condiciones. El trap `ERR` no es ejecutado si la instrucción fallida es parte de la lista de instrucciones inmediatamente después de una palabra clave `until` o `while`, parte de la condición que sigue a las palabras reservadas `if` o `elif`, parte de una instrucción ejecutada en una lista `&&` o `||` excepto la instrucción que sigue al último `&&` o `||`, cualquier instrucción en una tubería excepto la última o si el estado de retorno de la instrucción está siendo invertido usando `!`. Estas son las mismas condiciones cumplidas por la opción `errexit` (`-e`).

Las señales ignoradas en la entrada al intérprete no pueden ser atrapadas o eliminadas. Las señales atrapadas que no están siendo ignoradas son restablecidas a sus valores originales en un subintérprete o un entorno de subintérprete cuando se crea una.

El estado de retorno es cero a no ser que *sigspec* no especifique una señal válida.

`umask`

```
umask [-p] [-S] [modo]
```

Establece el proceso del intérprete de la máscara de creación de archivos a *modo*. Si *modo* comienza por un dígito, es interpretado como un número octal; si no, es interpretado como una máscara de modo simbólico similar a la aceptada por la instrucción `chmod`. Si se omite *modo*, se imprime el valor actual de la máscara. Si se proporciona la opción `-S` sin un argumento *modo*, la máscara se imprime en un formato simbólico. Si se proporciona la opción `-p`, y se omite *modo*, la salida es de forma que puede reusarse como entrada. El estado de retorno es cero si el modo se cambia exitosamente o si no se proporciona un argumento *modo*, y distinto a cero de no ser así.

Ten en cuenta que cuando el modo es interpretado como un número octal, cada número de umask es restado de `7`. Así, un umask de `022` resulta en permisos de `755`.

`unset`

```
unset [-fnv] [nombre]
```

Elimina cada variable o función *nombre*. Si se pasa la opción `-v`, cada*nombre* hace referencia a una variable del intérprete y se elimina dicha variable. Si pasa la opción `-f`, los *nombre*s se refieren a funciones del intérprete, y se elimina la definición de función. Si se proporciona la opción `-n` y *nombre* es una variable con el atributo *nameref*, *nombre* será desasignado en vez de la variable a la que referencia. `-n` no tiene efecto si se proporciona la opción `-f`. Si no se proporcionan opciones, cada *nombre* se refiere a una variable; si no hay variable con ese nombre, si lo hay. Las variables de solo lectura y las funciones no pueden ser desasignadas. Algunas variables del intérprete pierden su comportamiento especial cuando no están asignadas; este comportamiento se advierte en la descripción de las variables individuales. El estado de retorno es cero a no ser que *nombre* sea de solo lectura.

4.2 Instrucciones Integradas de Bash

Esta sección describe las instrucciones integradas que son únicas de o han sido extendidas en Bash. Algunas de estas instrucciones son especificadas en el estándar POSIX.

`alias`

```
alias [-p] [nombre[=valor] ...]
```

Sin argumentos o con la opción `-p`, `alias` imprime la lista de aliases en la salida estándar en una forma que permite que sean reutilizados como entrada. Si se proporcionan argumentos, se define un alias por cada *nombre* cuyo *valor* sea especificado. Si no se pasa un *valor*, se imprime el nombre y el valor del alias. Los aliases se describen en Sección 6.6 [Aliases], página 105,

`bind`

```
bind [-m mapa-de-teclas] [-lpsvPSVX]
bind [-m mapa-de-teclas] [-q función] [-u función]
     [-r secuencia-de-teclas]
bind [-m mapa-de-teclas] -f nombre-archivo
bind [-m mapa-de-teclas]
     -x secuencia-de-teclas:instrucción-de-intérprete
bind [-m mapa-de-teclas] secuencia-de-teclas:nombre-función
bind [-m mapa-de-teclas]
     secuencia-de-teclas:instrucción-readline
```

Muestra las asociaciones actuales de teclas y funciones Readline (véase Capítulo 8 [Edición en Línea de Órdenes], página 123), asocia una secuencia de teclas a una función o macro de Readline o establece una variable de Readline. Cada argumento que no es una opción es una instrucción tal como aparecería en un archivo de inicialización de Readline (véase Sección 8.3 [Archivo de Inicialización de Readline], página 126), pero cada asociación o

instrucción tiene que ser pasada como un argumento independiente; p. ej., '`"\C-x\C-r":re-read-init-file`'.

La opciones, si se proporcionan, tienen los siguientes significados:

`-m` *`mapa-de-teclas`*
: Usa *mapa-de-teclas* como el mapa de teclas afectado por las siguientes asociaciones. Son nombres aceptables de *mapa-de-teclas* `emacs`, `emacs-standard`, `emacs-meta`, `emacs-ctlx`, `vi`, `vi-move`, `vi-command` y `vi-insert`. `vi` es equivalente a `vi-command` (`vi-move` es también un sinónimo); `emacs` es equivalente a `emacs-standard`.

`-l`
: Lista los nombres de todas las funciones de Readline.

`-p`
: Muestra los nombres de las funciones y asociaciones de Readline de manera que puedan ser usadas como entrada o en una archivo de inicialización de Readline.

`-P`
: Lista los nombres de funciones y las asociaciones de Readline.

`-v`
: Muestra los nombres de variable y valores de Readline de una manera que puedan ser usados como entrada o en un archivo de inicialización de Readline.

`-V`
: Lista los nombres de variable y los valores actuales de Readline.

`-s`
: Muestra la secuencia de caracteres de Readline asociada a macros y las cadenas que producen de manera que pueda ser usada como entrada o en un archivo de inicialización Readline.

`-S`
: Muestra las secuencias de teclas de Readline asociadas a macros y las cadenas que producen.

`-f` *`nombre-de-archivo`*
: Lee asociaciones de teclas de *nombre-de-archivo*

`-q` *`función`*
: Consulta sobre las teclas que llaman a la *función* nombrada.

`-u` *`función`*
: Desasocia todas las teclas asociadas a la *función* nombrada.

`-r` *`secuencia-de-teclas`*
: Elimina cualquier asociación actual para *secuencia-de-teclas*

`-x` *`secuencia-de-teclas:instrucción-intérprete`*
: Hace que *instrucción-intérprete* se ejecute cada vez que se introduzca *sec-teclas*. Cuando se ejecuta *instrucción-intérprete* el intérprete asigna la variable `READLINE_LINE` con los contenidos del búfer de línea de Readline y las variables `READLINE_POINT` y `READLINE_MARK` con la ubicación actual del punto de inserción y el punto de inserción guardado (la *marca*), respectivamente. Si la instrucción ejecutada cambia el valor de `READLINE_LINE`, `READLINE_POINT` o `READLINE_MARK`, esos nuevos valores se verán reflejados en el estado de edición.

`-X` Lista todas las secuencias de teclas asociadas a instrucciones del intérprete y las instrucciones asociadas en un formato que puede ser reutilizado como entrada.

El estado de retorno es cero a no ser que se proporcione una opción inválida u ocurra un error.

`instrucción integrada`

`builtin [`*`instrucción-integrada`* `[`*`paráms`*`]]`

Ejecuta una instrucción integrada, pasándole *paráms*, y devuelve su estado de salida. Esto es útil cuando se define una función del intérprete con el mismo nombre que una función integrada del intérprete, conservando la funcionalidad de la instrucción integrada dentro de la función. El estado de retorno es distinto de cero si *instrucción-integrada* no es una instrucción integrada del intérprete.

`caller`

`caller [`*`expr`*`]`

Devuelve el contexto de cualquier llamada a una subrutina activa (una instrucción del intérprete o un guion ejecutado con las instrucciones integradas `.` o `source`).

Sin *expr*, `caller` muestra el número de línea y el nombre de archivo fuente de la actual llamada de subrutina. Si se proporciona un entero negativo como *expr*, `caller` muestra el número de línea, el nombre de subrutina y el archivo fuente correspondiente a esa posición en la actual pila de llamadas de ejecución. Esta información extra puede ser usada, por ejemplo, para imprimir un volcado de pila. El marco actual es el marco 0.

El valor de retorno es 0 a no ser que el intérprete no esté ejecutando una llamada de subrutina o *expr* no corresponda a una posición válida en la pila de llamadas.

`command`

`command [-pVv]` *`instrucción`* `[`*`argumentos`* `...]`

Ejecuta *instrucción* con *argumentos* ignorando cualquier instrucción del intérprete llamada *instrucción*. Solo se ejecutan las instrucciones integradas del intérprete o las instrucciones encontradas al inspeccionar el `PATH`. Si hay una instrucción del intérprete llamada `ls`, ejecutar '`command ls`' dentro de la función ejecutará la instrucción externa `ls` en vez de llamar a la función recursivamente. La opción `-p` equivale a usar un valor por defecto para `PATH` que garantiza encontrar todas las utilidades estándares. El estado de retorno en este caso es 127 si no se puede encontrar *instrucción* u ocurrió un error, y el estado de salida de *instrucción* en caso contrario.

Si se proporciona la opción `-V` o la `-v`, se imprime una descripción de *instrucción*. La opción `-v` hace que se muestre una única palabra indicando el nombre de la instrucción o archivo utilizado para llamar a *instrucción*; la opción `-V` produce una descripción más extensa. En este caso, el estado de retorno es cero si *instrucción* es encontrada, y distinto a cero en si no.

`declare`

`declare [-aAfFgiIlnrtux] [-p] [`*`nombre`*`[=`*`valor`*`] ...]`

Declara variables y les da atributos. Si no se pasan *nombre*s, entonces muestra los valores de las variables en su lugar.

La opción `-p` mostrará los atributos y valores de cada *nombre*. Cuando se usa la opción `-p` con argumentos *nombre*, se ignoran opciones adicionales, excepto `-f` y `-F`.

Cuando se proporciona `-p` sin argumentos *nombre*, `declare` mostrará los atributos y valores de todas las variables con los atributos especificados por las opciones adicionales. Si no se proporcionan otras opciones con `-p`, `declare` mostrará los atributos y valores de todas las variables del intérprete. La opción `-f` restringirá la salida a funciones del intérprete.

La opción `-F` inhibe la salida de las definiciones de funciones; solo se imprimen el nombre de función y los atributos. Si se habilita la opción del intérprete `extdebug` usando `shopt` (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73), se muestra también el archivo fuente y el número de línea donde cada *nombre* está definido. `-F` supone `-f`.

La opción `-g` obliga a que las variables sean creadas o modificadas en el alcance global, incluso cuando `declare` es ejecutado en una función del intérprete. Es ignorada en todos los demás casos.

La opción `-l` hace que las variables locales hereden los atributos (excepto el atributo *nameref*) y el valor de cualquier variable existente con el mismo *nombre* en un alcance que lo rodea. Si no hay una variable existente, la variable local está inicialmente deshabilitada.

Las siguientes opciones pueden ser usadas para restringir la salida a variables con los atributos especificados o para dar a las variables atributos:

`-a` Cada *nombre* es una variable de vector indexado (véase Sección 6.7 [Vectores], página 105).

`-A` Cada *nombre* es una variable de vector asociativo (véase Sección 6.7 [Vectores], página 105).

`-f` Usa solo nombres de función.

`-i` La variable será tratada como un entero; la evaluación aritmética (véase Sección 6.5 [Aritmética del Intérprete], página 103) se realiza cuando se asigna un valor a la variable.

`-l` Cuando se le asigna un valor a la variable, todos los caracteres en mayúscula son convertidos a minúscula. El atributo upper-case es deshabilitado.

`-n` Da a cada *nombre* el atributo *nameref*, convirtiéndolo en una referencia de nombre a otra variable. Esa otra variable está definida por el valor de *nombre*. Todas las referencias, asignaciones y modificaciones de atributo a *nombre*, excepto para aquellas que usan o cambian el mismo atributo `-n`, son realizadas en la variable referenciada por el valor *nombre*. El atributo nameref no puede ser aplicado a variables de vector.

`-r` Hace que *nombre*s sean de solo lectura. Después no se pueden asignar valores a estos nombres usando sentencias de asignación o eliminación.

`-t` Da a cada *nombre* el atributo `trace`. Las funciones rastreadas heredan las traps `DEBUG` y `RETURN` del intérprete ejecutor. El atributo trace no tiene ningún significado especial para variables.

`-u` Cuando se asigna un valor a una variable, todos los caracteres en minúscula son convertidos a mayúscula. El atributo lower-case es deshabilitado.

`-x` Marca cada *nombre* para exportar a posteriores instrucciones mediante el entorno.

Usar '`+`' en vez de '`-`' desactiva el atributo, con las excepciones de que '`+a`' y '`+A`' no pueden ser usados para destruir variables de vector y '`+r`' no eliminará el atributo readonly. Cuando se usa en una función, `declare` hace cada *nombre* local, como con la instrucción `local`, a no ser que se use la opción `-g`. Si un nombre de variable es seguido de =*valor*, se establece el valor de la variable a *valor*.

Cuando se usa `-a` o `-A` y la sintaxis de asignación compuesta para crear variables de vector, los atributos adicionales no tienen efecto hasta siguientes asignaciones.

El estado de retorno es cero a no ser que se encuentre una opción inválida, se realice un intento para definir una función usando '`-f foo=bar`', se realice un intento de asignar un valor a una variable de solo lectura, se realice un intento de asignar un valor a una variable de vector sin usar la sintaxis de asignaciones compuestas (véase Sección 6.7 [Vectores], página 105), uno de los *nombres* no sea un nombre de variable de intérprete válido, se realice un intento de desactivar el estado de solo lectura de una variable de solo lectura, se realice un intento de desactivar el estado de vector de una variable de vector o se realice un intento para mostrar una función inexistente con `-f`.

`echo`

```
echo [-neE] [arg ...]
```

Muestra los *arg*s, separados por espacios, terminado en una nueva línea. El estado de retorno es 0 a no ser que ocurra un error de escritura. Si se especifica `-n`, la nueva línea adicional se suprime. Si se pasa la opción `-e`, se habilita la interpretación de los siguientes caracteres escapados por barras invertidas. La opción `-E` deshabilita la interpretación de estos caracteres de escape, incluso en sistemas en los que son interpretados por defecto. La opción del intérprete `xpg_echo` puede ser usada para determinar dinámicamente si `echo` expande estos caracteres de escape por defecto o no. `echo` no interpreta `--` para indicar el final de las opciones.

`echo` interpreta las siguientes secuencias de escape:

`\a` alerta (timbre)

`\b` retroceso

`\c` suprime salida adicional

`\e`
`\E` escape

`\f` salto de página

`\n` nueva línea

`\r` retorno de carro

`\t` tabulación horizontal

`\v` tabulación vertical

`\\` barra invertida

`\0`*nnn* el carácter de ocho bits cuyo valor es el valor octal *nnn* (de cero a tres dígitos octales).

`\x`*HH* el carácter de ocho bits cuyo valor es el valor hexadecimal *HH* (uno o dos dígitos hexadecimales)

`\u`*HHHH* el carácter Unicode (ISO/IEC 10646) cuyo valor es el valor hexadecimal *HHHH* (de uno a cuatro dígitos hexadecimales)

`\U`*HHHHHHHH* el carácter Unicode (ISO/IEC 10646) cuyo valor es el valor hexadecimal *HHHHHHHH* (de uno a ocho dígitos hexadecimales)

`enable`

`enable [-a] [-dnps] [-f` *nombre-de-archivo*`] [`*nombre* `...]`

Habilita y deshabilita instrucciones integradas del intérprete. Deshabilitar una instrucción integrada permite que una instrucción del disco que tiene el mismo nombre que una instrucción integrada del intérprete sea ejecutada sin especificar un nombre de ruta completo, a pesar de que el intérprete normalmente busca instrucciones integradas antes que instrucciones del disco. Si se usa `-n`, se deshabilitan los *nombre*s. De lo contrario, son habilitados *nombre*s. Por ejemplo, para usar el binario `test` encontrado a través de `$PATH` en vez de la versión integrada del intérprete, escribe '`enable -n test`'.

Si se proporciona la opción `-p`, o no aparecen argumentos *nombre*, se imprime una lista de instrucciones del intérprete. Sin ningún otro argumento, la lista consta de todas las instrucciones habilitadas del intérprete. La opción `-a` equivale a listar cada instrucción integrada con una señal de si está o no habilitada.

La opción `-f` equivale a cargar la nueva instrucción integrada *nombre* del objeto compartido *nombre-de-archivo*, en sistemas que soportan carga dinámica. La opción `-d` borrará una instrucción integrada cargada con `-f`.

Si no hay opciones, se muestra una lista de todas las instrucciones integradas. La opción `-s` limita `enable` a las instrucciones integradas especiales POSIX. Si `-s` se usa con `-f`, la nueva instrucción integrada se convierte en una instrucción integrada especial (véase Sección 4.4 [Instrucciones Integradas Especiales], página 80).

El estado de retorno es cero a no ser que *nombre* no sea una instrucción integrada del intérprete o haya un error cargando una nueva instrucción integrada de un objeto compartido.

`help`

```
help [-dms] [patrón]
```

Muestra información útil sobre instrucciones integradas. Si se especifica *patrón*, `help` muestra información detallada sobre todas las instrucciones que coincidan con *patrón*, sino se imprime una lista de las instrucciones integradas.

La opciones, si se proporcionan, tienen los siguientes significados:

`-d` Muestra una descripción corta de cada *patrón*

`-m` Muestra la descripción de cada *patrón* en un formato tipo página-man

`-s` Muestra solo un resumen de uso corto por cada *patrón*

El estado de retorno es cero a no ser que ninguna instrucción coincida con *patrón*.

`let`

```
let expression [expresión ...]
```

La instrucción integrada `let` permite realizar operaciones aritméticas en variables del intérprete. Cada *expresión* se evalúa de acuerdo a las reglas detalladas adelante en Sección 6.5 [Aritmética del Intérprete], página 103. Si la última *expresión* evalúa a 0, `let` devuelve 1; de lo contrario, devuelve cero.

`local`

```
local [opción] nombre[=valor] ...
```

Para cada argumento se crea una variable local llamada *nombre*, y se le asigna *valor*. La *opción* puede ser cualquiera de las opciones aceptadas por `declare`. `local` puede usarse solo dentro de una función; hace que la variable *nombre* tenga un alcance visible restringido a esa función y a sus hijos. Si *nombre* es '`-`', el conjunto de opciones del intérprete se hacen locales para la función en que se llama a `local`: las opciones del intérprete cambiadas usando la instrucción integrada `set` dentro de la función son restablecidas a sus valores originales cuando retorna la función. La restauración surge efecto como si se hubieran ejecutado una serie de instrucciones `set` para restaurar los valores que estaban en su lugar antes de la función. El estado de retorno es cero a no ser que `local` se use fuera de una función, se proporcione un *nombre* inválido o *nombre* sea una variable de solo lectura.

`logout`

```
logout [n]
```

Sale de un intérprete de acceso, devolviendo un estado de *n* al padre del intérprete.

`mapfile`

```
mapfile [-d delim] [-n cuenta] [-O origen] [-s cuenta]
```

```
[-t] [-u da] [-C retrollamada] [-c cuantía] [vector]
```

Lee líneas de la entrada estándar en la variable de vector indexado *vector*, o del descriptor de archivo *da* si se proporciona la opción `-u`. La variable `MAPFILE` es el *vector* predeterminado. Si se proporcionan, las opciones tienen el siguiente significado:

`-d` Se usa el primer carácter de *delim* para terminar cada línea de entrada, en vez de nueva línea. Si *delim* es la cadena vacía, `mapfile` terminará una línea cuando lea un carácter NUL.

`-n` Copia como máximo *cuenta* líneas. Si *cuenta* es 0, se copian todas las líneas.

`-O` Empieza asignando a *vector* en el índice *origen*. El índice predeterminado es 0.

`-s` Descarta las primeras *cuenta* líneas leídas.

`-t` Elimina un *delim* al final (nueva línea predeterminada) de cada línea leída.

`-u` La líneas son leídas del descriptor de archivo *da* en vez de la entrada estándar.

`-C` Evalúa *retrollamada* cada vez que *cuantía* líneas son leídas. La opción `-c` especifica *cuantía*.

`-c` Especifica el número de líneas leídas entre cada llamada a *retrollamada*.

Si `-C` se proporciona sin `-c`, la cuantía por defecto es 5000. Cuando se evalúa *retrollamada*, es proporcionado el índice del siguiente elemento vector a ser asignado y la línea a ser asignada a ese elemento como argumentos adicionales. *retrollamada* se evalúa después de que la línea sea leída, pero antes de que el elemento vector sea asignado.

Si no se proporciona con un origen explícito, `mapfile` liberará el *vector* antes de asignar a él.

`mapfile` retorna exitosamente a no ser que se proporcione una opción inválida, *vector* sea inválido o inasignable o *vector* no sea un vector indexado.

`printf`

```
printf [-v var] formato [argumentos]
```

Escribe los *argumentos* formateados en la salida estándar bajo el control del *format*. La opción `-v` hace que la salida sea asignada a la variable *var* en vez de ser imprimida a la salida estándar.

El *format* es una cadena de caracteres que contiene tres tipos de objetos: caracteres sencillos, que son simplemente copiados a la salida estándar; secuencias de caracteres de escape, que son convertidos y copiados a la salida estándar; y especificaciones de formato, cada una de las cuales provoca la impresión del siguiente *argumento* sucesivo. Además de los formatos estándares de `printf(1)`, `printf` interpreta las siguientes extensiones:

`%b` Hace que `printf` expanda las secuencias de barras invertidas de escape en el *argumento* correspondiente de la misma forma que `echo -e` (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).

`%q` Hace que `printf` muestre el *argumento* correspondiente en un formato que puede ser reusado como entrada del intérprete.

`%(`*datefmt*`)T`
Hace que `printf` muestre la cadena fecha-tiempo resultante de usar *datefmt* como una cadena de formato para `strftime`(3). El *argumento* correspondiente es un entero que representa el número de segundos desde la época. Se pueden usar dos argumentos especiales: -1 representa el tiempo actual, y -2 representa el tiempo en que el intérprete fue invocado. Si no se especifican parámetros, la conversión se comporta como si se le hubiera pasado -1. Esto es una excepción al comportamiento usual de `printf`.

Todas las directivas %b, %q y %T usan el campo de anchura y los argumentos de precisión de la especificación de formato y escriben esos tantos bytes desde (o usan ese campo de anchura para) el argumento expandido, que normalmente contiene más caracteres que el original.

Los argumentos para especificadores de formato que no son cadenas son tratados como constantes del lenguaje C, excepto que se permite un signo de mayor o menor como prefijo, y si el carácter prefijado es una comilla simple o doble, el valor es el valor ASCII del siguiente carácter.

El *formato* es reusado según sea necesario para consumir todos los *argumentos*. Si el *format* requiere más *parámetros* que los proporcionados, las especificaciones extra de formato se comportan como si un valor cero u nulo, según la conveniencia, hubiera sido proporcionado. El valor de retorno es cero en caso de éxito, distinto a cero en caso de fallo.

`read`

```
read [-ers] [-a nombrev] [-d delim] [-i texto] [-n ncaracts]
    [-N ncaracts] [-p prompt] [-t tiempo-límite] [-u da] [nom-
bre ...]
```

Se lee una línea de la entrada estándar, o del descriptor de archivo *da* proporcionado como argumento a la opción `-u`, se divide en palabras como se describe más arriba en Sección 3.5.7 [División de Palabras], página 34, y la primera palabra se asigna al primer *nombre*, la segunda palabra al segundo *nombre* y demás. Si hay más palabras que nombres, las palabras restantes y los delimitadores que intervienen se asignan al último *nombre*. Si hay menos palabras leídas del flujo de entrada que nombres, el resto de nombres se asignan a valores vacíos. Los caracteres en el valor de la variable `IFS` se usan para dividir la línea en palabras usando las mismas reglas que el intérprete usa para la expansión (descritas anteriormente en Sección 3.5.7 [División de Palabras], página 34). El carácter de barra invertida '\' puede usarse para eliminar cualquier significado especial para el siguiente carácter leído y para la continuación de línea.

La opciones, si se proporcionan, tienen los siguientes significados:

`-a` *nombrev*
: Las palabras son asignadas a los índices secuenciales de la variable de vector *nombrev*, empezando por 0. Todos los elementos son eliminados de *nombrev* antes de la asignación. Otros argumentos *nombre* son ignorados.

`-d` *delim*
: Se usa el primer carácter de *delim* para terminar la línea de entrada, en vez de nueva línea. Si *delim* es la cadena vacía, `read` terminará una línea cuando lea un carácter NUL.

`-e`
: Readline (véase Capítulo 8 [Edición en Línea de Órdenes], página 123) se usa para obtener la línea. Readline usa los ajustes de edición actuales (o predeterminados, si la edición de línea no estaba previamente activa), pero usa la compleción de nombre de archivo predeterminada de Readline.

`-i` *texto*
: Si Readline está siendo usado para leer la línea, *texto* es colocado en el búfer de edición entes de que la edición comience.

`-n` *ncaracts*
: `read` retorna después de leer *ncaracts* en vez de esperar a una línea de entrada completa, pero atiende al delimitador si son leídos menos de *ncaracts* antes del delimitador.

`-N` *ncaracts*
: `read` retorna después de leer exactamente *ncaracts* caracteres en vez de esperar a una línea completa de entrada, a no ser que se encuentre EOF o `read` se quede sin tiempo. Los caracteres de delimitadores encontrados en la entrada no son tratados de forma especial y no hacen que `read` retorne hasta que son leídos *ncaracts* caracteres. El resultado no es dividido en los caracteres en `IFS`; la intención es que sean asignados a la variable exactamente los caracteres leídos (con la excepción de la barra invertida; consulta la opción `-r` abajo).

`-p` *prompt*
: Muestra *prompt*, sin una nueva línea al final, antes de tratar de leer alguna entrada. El prompt se muestra solo si la entrada procede de una terminal.

`-r`
: Si se pasa esta opción, la barra invertida no actúa como un carácter de escape. Se considera que la barra invertida es parte de la línea. En concreto, una pareja barra invertida-nueva línea no puede ser usada entonces como una continuación de línea.

`-s`
: Modo silencioso. Si la entrada procede de una terminal, no muestra los caracteres escritos.

`-t` *tiempo-límite*
: Hace que `read` deje de esperar y devuelve un fallo si una línea completa de entrada (o un número especificado de caracteres) no es

leído dentro de *tiempo-límite* segundos. *tiempo-límite* puede ser un número decimal con una parte fraccional que sigue al punto decimal. Esta opción solo es efectiva si `read` está leyendo entrada de una terminal, tubería u otro archivo especial; no tiene efecto durante la lectura de archivos normales. Si a `read` se le agota el tiempo, `read` guarda cualquier entrada leída en la variable especificada *nombre*. Si *tiempo-límite* es 0, `read` retorna inmediatamente, sin tratar de leer ningún dato. El estado de salida es 0 si la entrada está disponible en el descriptor de archivo especificado, distinto de cero de lo contrario. El estado de salida es mayor que 128 si se excede el tiempo límite.

`-u` *da* Lee entrada del descriptor de archivo *da*.

Si no se proporcionan *nombre*s, la línea leída, sin el delimitador de fin opcional pero aparte de eso sin modificar, se asigna a la variable `REPLY`. El estado de salida es cero, a no ser que se llegue al final de archivo, `read` se quede sin tiempo (en cuyo caso el estado es mayor que 128), ocurra un error de asignación de variable (como asignar a una variable de solo lectura) o se proporcione un descriptor de archivo inválido como el argumento de `-u`.

`readarray`

```
readarray [-d delim] [-n cuenta] [-O origen] [-s cuenta] [-t] [-u da]
    [-C retrollamada] [-c cuantía] [vector]
```

Lee líneas de la entrada estándar en la variable de vector indexada *vector*, o del descriptor de archivo *da* si se proporciona la opción `-u`.

Un sinónimo de `mapfile`.

`source`

```
source nombre-de-archivo
```

Un sinónimo de `.` (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48).

`type`

```
type [-afptP] [nombre ...]
```

Por cada *nombre*, indica como sería interpretado si es usado como nombre de instrucción.

Si se usa la opción `-t`, `type` imprime una solo palabra que es una de '`alias`', '`function`', '`builtin`', '`file`' o '`keyword`', si *nombre* es un alias, función del intérprete, instrucción integrada del intérprete, archivo de disco o palabra reservada del intérprete respectivamente. Si no se encuentra el *nombre*, no se imprime nada y `type` devuelve un estado de fallo.

Si se usa la opción `-p`, `type` devuelve o el nombre del archivo de disco que sería ejecutado o nada si `-t` no devolviera '`file`'.

La opción `-P` fuerza una búsqueda de ruta por cada *nombre*, incluso si `-t` no devolviera '`file`'.

Si se crea el resumen criptográfico de una instrucción, `-p` y `-P` imprimen el valor del resumen criptográfico creado, que no es necesariamente el archivo que aparece primero en `$PATH`.

Si se usa la opción `-a`, `type` devuelve todos los lugares que contienen un ejecutable llamado *file*. Esto incluye aliases y funciones, solo si la opción `-p` no se usa también.

Si se usa la opción `-f`, `type` no intenta buscar instrucciones del intérprete, como con la instrucción integrada `command`.

El estado de retorno es cero si se encuentran todos los *nombres*, distinto a cero si alguno no es encontrado.

`typeset`

```
typeset [-afFgrxilnrtux] [-p] [nombre[=valor] ...]
```

La instrucción `typeset` se proporciona por compatibilidad con el intérprete Korn. Es un sinónimo de la instrucción integrada `declare`.

`ulimit`

```
ulimit [-HS] -a
ulimit [-HS] [-bcdefiklmnpqrstuvxPRT] [límite]
```

`ulimit` proporciona control sobre los recursos disponibles para los procesos iniciados por el intérprete, en sistemas que permiten tal control. Si se pasa una opción, se interpreta de la siguiente forma:

`-S` Cambia e informa el límite suave asociado con un recurso.

`-H` Cambia e informa sobre el límite duro asociado con un recurso.

`-a` Se informa de todos los límites actuales; no establece límites.

`-b` El tamaño máximo de búfer de socket.

`-c` El tamaño máximo de archivos de núcleo creados.

`-d` El tamaño máximo de un segmento de datos de proceso.

`-e` La prioridad máxima de planificación ("nice").

`-f` El tamaño máximo de archivos escritos por el intérprete y sus hijos.

`-i` El tamaño máximo de señales pendientes.

`-k` El número máximo de kqueues que pueden ser asignadas.

`-l` El máximo tamaño que puede ser bloqueado en memoria.

`-m` El máximo tamaño residente establecido (muchos sistemas no respetan este límite).

`-n` El máximo número de descriptores de archivo abiertos (la mayoría de sistemas no permiten que este valor sea establecido)

`-p` El tamaño del búfer de la tubería.

`-q` El número máximo de bytes en las colas de mensaje POSIX.

`-r` La máxima prioridad de planificación en tiempo real.

`-s` El tamaño máximo de pila.

`-t` La cantidad máxima de tiempo de cpu en segundos.

`-u` El máximo número de procesos disponibles para un único usuario.

`-v` La máxima cantidad de memoria virtual disponible para el intérprete, y, en algunos sistemas, para sus hijos.

`-x` El máximo número de bloqueos de archivo.

`-P` El máximo número de pseudoterminales.

`-R` El tiempo máximo que un proceso en tiempo real puede ejecutarse antes de bloquearse, en milisegundos.

`-T` El máximo número de procesos.

Si se especifica *límite* y no se usa la opción `-a`, *límite* es el nuevo valor del recurso especificado. Los valores *límite* especiales `hard`, `soft` y `unlimited` equivalen respectivamente al límite duro actual, el límite blando actual y no límite respectivamente. Un límite duro no puede aumentarse por un usuario no root una vez que haya sido asignado; un límite blando puede ser aumentado hasta el valor del límite duro. De lo contrario, el valor actual del límite blando para el recurso especificado se muestra por pantalla, a no ser que se proporcione la opción `-H`. Cuando se especifica más de un recurso, el nombre del límite y la unidad, si es apropiado, se muestran por pantalla antes del valor. Al establecer nuevos límites, si no se proporcionan ni `-H` ni `-S`, se asignan tanto el límite duro como el blando. Si no se proporcionan opciones, se asume `-f`. Los valores son en incrementos de 1024 bytes. excepto para `-t`, cuyos valores son en segundos; `-R`, que es en milisegundos; `-p`, que es en unidades de bloques de 512 bytes; `-P`, `-T`, `-b`, `-k`, `-n` and `-u`, que son valores sin escala; y, en Modo POSIX (véase Sección 6.11 [Modo POSIX de Bash], página 111), `-c` y `-f`, que son en incrementos de 512 bytes.

El estado de retorno es cero a no ser que se proporcione un argumento u opción inválida, u ocurra un error al establecer un nuevo límite.

`unalias`

```
unalias [-a] [nombre ... ]
```

Elimina cada *nombre* de la lista de aliases. Si se proporciona `-a`, todos lo aliases son eliminados. Los aliases son descritos en Sección 6.6 [Aliases], página 105.

4.3 Modificando el Comportamiento del Intérprete

4.3.1 La Instrucción Integrada Set

Esta instrucción integrada es tan complicada que se merece su propia sección. `set` te permite cambiar los valores de las opciones del intérprete y establecer los parámetros posicionales, o mostrar los nombres y valores de las variables del intérprete.

`set`

```
set [--abefhkmnptuvxBCEHPT] [-o nombre-de-opción]
```

```
    [argumento ...]
set [+abefhkmnptuvxBCEHPT] [+o nombre-de-opción]
    [argumento ...]
```

Si no se proporcionan opciones o argumentos, `set` muestra los nombres y valores de todas las variables y funciones del intérprete, ordenadas según la actual configuración regional, en un formato que puede ser reutilizado como entrada para establecer o eliminar las variables actualmente establecidas. Las variables de solo lectura no pueden ser eliminadas. En modo POSIX, solo son listadas las variables del intérprete.

Cuando se proporcionan opciones, establecen o eliminan los atributos del intérprete. Las opciones, si se especifican, tienen los siguientes significados:

`-a` A cada variable o función que es creada o modificada se le da el atributo de exportación y es marcada para exportar al entorno de las posteriores instrucciones.

`-b` Hace que se informe inmediatamente del estado de las tareas en segundo plano terminadas, en vez de imprimir antes del siguiente prompt primario.

`-e` Sale del intérprete inmediatamente si una tubería (véase Sección 3.2.3 [Tuberías], página 9), que puede consistir en una única instrucción simple (véase Sección 3.2.2 [Instrucciones Simples], página 8), una lista (véase Sección 3.2.4 [Listas], página 10) o una instrucción compuesta (véase Sección 3.2.5 [Instrucciones Compuestas], página 10) devuelve un estado distinto de cero. El intérprete no se cierra si la instrucción que falla es parte de la lista de instrucciones que sigue inmediatamente una palabra clave `while` o `until`, parte de una comprobación en un oración `if`, parte de cualquier instrucción ejecutada en una lista `&&` o `||` excepto la instrucción que sigue al último `&&` o `||`, cualquier instrucción en una tubería excepto la última, o si el estado de retorno es invertido con `!`. Si una instrucción compuesta distinta de un subintérprete retorna un estado distinto a cero porque una instrucción falló mientras se ignoraba `-e`, no se cierra el intérprete. Se ejecuta una trap en `ERR`, si está establecida, antes de que el intérprete salga.

Esta opción se aplica al entorno del intérprete y a cada entorno de subintérprete de forma separada (véase Sección 3.7.3 [Entorno de Ejecución de Instrucciones], página 43), y puede provocar que los subintérpretes se cierren antes de ejecutar todas las instrucciones en el subintérprete.

Si una instrucción compuesta o una función del intérprete se ejecuta en un contexto en el que `-e` esté siendo ignorado, ninguna de las instrucciones ejecutadas dentro de la instrucción compuesta o cuerpo de la función se verá afectada por el ajuste `-e`, incluso si `-e` está establecido y una instrucción devuelva un estado de fallo. Si una instrucción compuesta o función del intérprete establece `-e`

es ignorado, ese ajuste no tendrá ningún efecto hasta que la instrucción compuesta o la instrucción que contiene la llamada a la función se complete.

`-f` Deshabilita la expansión de nombre de archivo (globbing).

`-h` Ubica y recuerda instrucciones (resumen criptográfico) mientras son consultadas para su ejecución. Esta opción está habilitada por defecto.

`-k` Todos los argumentos en forma de sentencias de asignación son ubicados en el entorno para una instrucción, no solo aquellos que preceden al nombre de la instrucción.

`-m` Es habilitado el control de tareas (véase Capítulo 7 [Control de Tareas], página 118). Todos los procesos se ejecutan en un grupo de procesos separado. Cuando se completa una tarea en segundo plano, el intérprete imprime una línea que contiene su estado de salida.

`-n` Lee instrucciones pero no las ejecuta. Esto puede ser utilizado para comprobar un guion en busca de errores de sintaxis. Esta opción es ignorada por los intérpretes interactivos.

`-o` *nombre-de-opción*
Establece la opción correspondiente a *nombre-de-opción*:

`allexport`
Equivalente a `-a`.

`braceexpand`
Equivalente a `-B`.

`emacs` Usa una interfaz de edición de línea (véase Capítulo 8 [Edición en Línea de Órdenes], página 123) estilo `emacs`. Esto también afecta a la interfaz de edición usada para `read -e`.

`errexit` Equivalente a `-e`.

`errexit` Equivalente a `-E`.

`functrace`
Equivalente a `-T`.

`hashall` Equivalente a `-h`.

`histexpand`
Equivalente a `-H`.

`history` Habilita el historial de instrucciones, como se describe en Sección 9.1 [Servicios del Historial de Bash], página 161. Esta opción está activada por defecto en intérpretes interactivos.

`ignoreeof`
No se saldrá de un intérprete interactivo al leer EOF.

`keyword` Equivalente a `-k`.

`monitor` Equivalente a `-m`.

`noclobber`
Equivalente a `-C`.

`noexec` Equivalente a `-n`.

`noglob` Equivalente a `-f`.

`nolog` Actualmente ignorado.

`notify` Equivalente a `-b`.

`nounset` Equivalente a `-u`.

`onecmd` Equivalente a `-t`.

`physical` Equivalente a `-P`.

`pipefail` Si se habilita, el valor de retorno de una tubería es el valor de la última (la más a la derecha) instrucción que finalice con un estado distinto de cero, o cero si todas las instrucciones en la tubería finalizan con éxito. Esta opción está deshabilitada por defecto.

`posix` Cambia el comportamiento de Bash donde la operación por defecto difiere del estándar POSIX para cumplir el estándar (véase Sección 6.11 [Modo POSIX de Bash], página 111). Esto está pensado para hacer que Bash se comporte como un componente preciso de ese estándar.

`privileged`
Equivalente a `-p`.

`verbose` Equivalente a `-v`.

`vi` Usa la interfaz de edición de línea estilo `vi`. Esto también afecta a la interfaz de edición usada por `read -e`.

`xtrace` Equivalente a `-x`.

`-p` Activa el modo privilegiado. En este modo los archivos `$BASH_ENV` y `$env` no son procesados, las funciones no son heredadas del entorno y las variables `SHELLOPTS`, `BASHOPTS`, `CDPATH` y `GLOBIGNORE`, si aparecen en el entorno, son ignoradas. Si el intérprete se inicia con el identificador efectivo del usuario (grupo) diferente del identificador del usuario (grupo) real y no se proporciona la opción `-p`, se llevan estas acciones, y el id efectivo del usuario se establece al id real del usuario. Si la opción `-p` se proporciona durante el inicio, no se restablece el id de usuario efectivo. Desactivar esta opción hace que los identificadores del usuario y el grupo efectivos sean establecidos a los identificadores del usuario y el grupo reales.

`-t` Sale después de leer y ejecutar una instrucción.

`-u` Trata las variables y parámetros sin establecer que sean distintos a '`@`' o '`*`' como un error cuando se realiza la expansión de parámetros. Se escribirá un mensaje de error a la salida de error estándar, y un intérprete no interactivo se cerrará.

`-v` Imprime las líneas de entrada del intérprete según son leídas.

`-x` Imprime un rastro de instrucciones simples, instrucciones `for`, instrucciones `case`, instrucciones `select` y instrucciones aritméticas `for` y sus argumentos o listas de palabras asociadas después de que sean expandidos y antes de que sean ejecutados. El valor de la variable `PS4` es expandido, y el valor resultante es imprimido antes de la instrucción y sus argumentos expandidos.

`-B` El intérprete realizará la expansión de llaves (véase Sección 3.5.1 [Expansión de Llaves], página 25). Está opción está activada por defecto.

`-C` Evita que la redirección de salida usando '`>`', '`>&`' y '`<>`' sobreescriba archivos existentes.

`-E` Si está establecida, cualquier trap en `ERR` es heredada por funciones del intérprete, sustituciones de intrucciones e intrucciones ejecutadas en un entorno de subintérprete. La trap `ERR` no se hereda normalmente en tales casos.

`-H` Habilita la sustitución de historial estilo '`!`' (véase Sección 9.3 [Interacción con el Historial], página 163). Está opción está activada por defecto en intérpretes interactivos.

`-P` Si está habilitada, no resuelve enlaces simbólicos al realizar instrucciones como `cd` que cambian el directorio actual. Se usa el directorio físico en su lugar. Por defecto, Bash sigue la cadena lógica de directorios cuando realiza instrucciones que cambian el directorio actual.

Por ejemplo, si `/usr/sys` es un enlace simbólico a `/usr/local/sys`, entonces:

```
$ cd /usr/sys; echo $PWD
/usr/sys
$ cd ..; pwd
/usr
```

Si `set -P` está activada, entonces:

```
$ cd /usr/sys; echo $PWD
/usr/local/sys
$ cd ..; pwd
/usr/local
```

`-T` Si está establecida, cualquier trap en `DEBUG` y `RETURN` son heredadas por funciones del intérprete, sustituciones de instrucciones e instrucciones ejecutadas en un entorno de subintérprete. Las trampas `DEBUG` y `RETURN` no son heredadas normalmente en tales casos.

`--` Si ningún argumento sigue a esta opción, los parámetros posicionales son eliminados. En caso contrario, los parámetros posicionales son establecidos a *argumentos*, incluso si algunos de ellos comienzan por '`-`'.

`-` Señala el fin de las opciones, hace que todos los *argumentos* restantes sean asignados a los parámetros posicionales. Las opciones `-x` y `-v` son desactivadas. Si no hay argumentos, los parámetros posicionales no cambian.

Usar '`+`' en vez de '`-`' hace que estás opciones se desactiven. Las opciones también pueden ser usadas al invocar al intérprete. El conjunto actual de opciones se puede encontrar en `$-`.

Los N *argumentos* porsicionales restantes son parámetros posicionales y son asignados, en orden, a `$1`, `$2`, ... `$N`. El parámetro especial `#` es establecido a N.

El estado de retorno siempre es cero a no ser que se proporcione una opción inválida.

4.3.2 La Instrucción Integrada Shopt

Esta instrucción integrada te permite cambiar el comportamiento adicional del intérprete.

`shopt`

```
shopt [-pqsu] [-o] [nombre-de-opción ...]
```

Alterna los valores de los ajustes que controlan el comportamiento opcional del intérprete. Los ajustes pueden ser los listados a continuación o, si se usa la opción `-o`, aquellos disponibles con la opción `-o` para la instrucción integrada `set` (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68). Sin opciones, o con la opción `-p`, se muestra una lista de todas las opciones que pueden ser habilitadas, con una indicación de si está habilitada o no; si se proporcionan *nombres-de-opciones*, se limita la salida a esas opciones. La opción `-p` hace que la salida se muestre de forma que pueda ser reutilizada como entrada. Las otras opciones tienen los siguientes significados:

`-s` Activa (habilita) cada *nombre-de-opción*.

`-u` Desactiva (deshabilita) cada *nombre-de-opción*.

`-q` Suprime la salida normal; el estado de retorno indica si el *nombre-de-opción* está habilitado o deshabilitado. Si se pasan varios argumentos *nombre-de-opción* con `-q`, el estado de retorno es cero si todos los *nombres-de-opción* están habilitados; distinto de cero en caso contrario.

`-o` Limita los valores de *nombre-de-opción* a aquellos definidos por la opción `-o` para la instrucción integrada `set` (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68).

Si `-s` o `-u` se usan sin argumentos *nombre-de-opción*, `shopt` muestra solo aquellas opciones que están habilitadas o deshabilitadas, respectivamente.

A no ser que se indique lo contrario, las opciones `shopt` están deshabilitadas (off) por defecto.

El estado de retorno al listar opciones es cero si todos los *nombres-de-opciones* están habilitados, diferente a cero en caso contrario. Al habilitar o deshabilitar opciones, el estado de retorno es cero a no ser que *nombre-de-opción* no sea una opción del intérprete válida.

La lista de opciones `shopt` es:

`assoc_expand_once`
: Si está habilitada, el intérprete suprime la evaluación múltiple de superíndices de vectores durante la evaluación de una expresión aritmética, mientras ejecuta instrucciones integradas que pueden realizar asignaciones de variables y mientras ejecuta instrucciones integradas que realizan la desreferenciación de vectores.

`autocd`
: Si está habilitada, un nombre de instrucción que es el nombre de un directorio es ejecutado como si fuera el argumento de la instrucción `cd`. Esta opción solo es usada por intérpretes interactivos.

`cdable_vars`
: Si esto está habilitado, se asume que un argumento para la instrucción integrada `cd` que no es un directorio es el nombre de una variable cuyo valor es el directorio al que cambiar.

`cdspell`
: Si está habilitada, serán corregidos los errores de escritura menores de un componente de directorio en una instrucción `cd`. Los errores comprobados son caracteres transpuestos, y un carácter de más. Si se encuentra una corrección, se imprime la ruta corregida, y la instrucción procede. Esta opción solo es usada por intérpretes interactivos.

`checkhash`
: Si esto está habilitado, Bash comprueba que una instrucción encontrada en la tabla hash existe antes de intentar ejecutarla. Si una instrucción con un hash ya no existe, se realiza una búsqueda de ruta normal.

`checkjobs`
: Si está habilitada, Bash lista el estado de cualquier tarea detenida o en ejecución antes de que salga de un intérprete interactivo. Si alguna tarea está en ejecución, esto hace que la salida sea pospuesta hasta que se intente una segunda salida sin una instrucción interviniente (véase Capítulo 7 [Control de Tareas], página 118). El intérprete siempre pospone la salida si se detiene cualquier tarea.

`checkwinsize`
: Si está habilitada, Bash comprueba la tamaño de la ventana después de cada instrucción externa (no integrada) y, si es necesario, actualiza los valores de `LINES` y `COLUMNS`. Esta opción está habilitada por defecto.

`cmdhist` Si está habilitada, Bash intenta guardar todas las líneas de una instrucción de varias líneas en la misma entrada del historial. Esto permite una reedición sencilla de instrucciones de varias líneas. Esta opción está habilitada por defecto, pero solo tiene efecto si el historial de instrucciones está habilitado (véase Sección 9.1 [Servicios del Historial de Bash], página 161).

`compat31`
`compat32`
`compat40`
`compat41`
`compat42`
`compat43`
`compat44` Estos controlan aspectos del modo de compatibilidad del intérprete (véase Sección 6.12 [Modo de Compatibilidad del Intérprete], página 115).

`complete_fullquote`
Si está habilitada, Bash entrecomilla todos los metacaracteres del intérprete en nombres de archivo y nombre de directorio al realizar la compleción. Si no está habilitada, Bash elimina metacaracteres como el símbolo de dólar del conjunto de caracteres que será entrecomillado en los nombres de archivo completados cuando estos metacaracteres aparezcan en referencias de variables del intérprete en palabras que deben ser completadas. Esto significa que los símbolos de dólar en nombres de variable que expanden a directorios no serán entrecomillados; sin embargo, cualquier signo de dólar que aparezca en nombres de archivo no será entrecomillado, tampoco. Esto está activado solo cuando bash está usando barras invertidas para entrecomillar nombres de archivo completados. Esta variable está habilitada por defecto, que es el comportamiento habitual de Bash en versiones a partir de la 4.2.

`direxpand`
Si está establecida, Bash reemplaza los nombres de directorio con los resultados de la expansión de palabra al realizar la compleción de nombre de archivo. Esto cambia los contenidos del búfer de edición de readline. Si no está establecida, Bash trata de preservar lo que el usuario tecleó.

`dirspell` Si está establecida, Bash intenta la corrección de escritura en nombres de directorios durante la compleción de palabras si el nombre del directorio proporcionado inicialmente no existe.

`dotglob` Si está establecida, Bash incluye los nombres que empiezan por un '.' en los resultados de la expansión de nombre de archivo. Los nombres de archivo '`.`' y '`..`' siempre tienen que ser coincididos explícitamente, incluso si está habilitada `dotglob`.

`execfail` Si esto está establecido, un intérprete no interactivo no se cerrará si no puede ejecutar el archivo especificado como un argumento para

la instrucción integrada `exec`. No se cierra un intérprete interactivo si `exec` falla.

`expand_aliases`
Si está establecida, los aliases son expandidos como se describe abajo en Aliases, Sección 6.6 [Aliases], página 105. Esta opción está habilitada por defecto para intérpretes interactivos.

`extdebug` Si se asigna durante la llamada al intérprete, o en un archivo de inicio del intérprete, se dispone a ejecutar el perfil del depurador antes de que el intérprete se inicie, idéntica a la opción `--debugger`. Si se asigna después de su llamada, se activa el comportamiento pensado para usarse por depuradores:

1. La opción `-F` para la instrucción integrada `declare`(véase Sección 4.2 [Instrucciones Integradas de Bash], página 56) muestra el nombre de archivo fuente y el número de línea correspondiente a cada nombre de función proporcionado como argumento.
2. Si la instrucción ejecutada por la trap `DEBUG` devuelve un valor distinto a cero, se salta el siguiente comando y no se ejecuta.
3. Si la instrucción ejecutada por la trap `DEBUG` devuelve un valor de 2, y el intérprete se está ejecutando en una subrutina (una función del intérprete o un guion del intérprete ejecutado por las instrucciones integradas `.` o `source`), el intérprete simula una llamada a `return`.
4. `BASH_ARGC` y `BASH_ARGV` son actualizados según lo descrito en sus descripciones (véase Sección 5.2 [Variables de Bash], página 82).
5. Se habilita el rastreo de funciones: la sustitución de instrucciones, las funciones del intérprete y los subintérpretes invocados con `( instrucción )` heredan las trap `DEBUG` y `RETURN`.
6. El rastreo de errores es habilitado: la sustitución de instrucciones, las funciones del intérprete y los subintérpretes invocados con `( instrucción )` heredan la trap `ERR`.

`extglob` Si está habilitada, son habilitadas las funcionalidades extendidas de coincidencia de patrones descritas más arriba (véase Sección 3.5.8.1 [Coincidencia de Patrones], página 36).

`extquote` Si está habilitada, el entrecomillado `$'cadena'` and `$"cadena"` es realizado dentro de expresiones `${parámetro}` encerradas en comillas dobles. Esta opción está habilitada por defecto.

`failglob` Si está habilitada, los patrones que no consigan coincidir con nombres de archivo durante la expansión de nombre de archivo producen un error de expansión.

`force_fignore`
Si está habilitada, los sufijos especificados por la variable del intérprete `FIGNORE` hacen que las palabras sean ignoradas al

realizar la compleción de palabras incluso si las palabras ignoradas son las únicas compleciones posibles. Véase Sección 5.2 [Variables de Bash], página 82, para una descripción de `FIGNORE`. Esta opción está habilitada por defecto.

`globasciiranges`
: Si está habilitada, las expresiones de rango usadas en las expresiones de llaves de coincidencia de patrones (véase Sección 3.5.8.1 [Coincidencia de Patrones], página 36) se comportan como en la configuración regional C tradicional al realizar comparaciones. Es decir, no se tiene en cuenta la actual secuencia de ordenación de configuración regional, por lo que '`b`' no se ordena entre '`A`' y '`B`', y los caracteres ASCII en minúscula o mayúscula se ordenarán juntos.

`globstar`
: Si está habilitada, el patrón '`**`' usado en un contexto de expansión de nombre de archivo corresponderá a todos los archivos y cero o más directorios y subdirectorios. Si al patrón lo sigue un '`/`', solo coinciden directorios y subdirectorios.

`gnu_errfmt`
: Si está habilitada, los mensajes de error del intérprete son escritos en el formato de error estándar de GNU.

`histappend`
: Si está habilitada, la lista del historial es añadida al nombre de archivo nombrado por el valor de la variable `HISTFILE` cuando el intérprete se cierra, en vez de sobreescribir el archivo.

`histreedit`
: Si está habilitada, y se está usando Readline, se le da la oportunidad al usuario de reeditar una sustitución del historial fallida.

`histverify`
: Si está habilitada, y se está usando Readline, los resultados de la sustitución del historial no son inmediatamente pasados al analizador del intérprete. En su lugar, la línea resultante es cargada en el búfer de edición de Readline, permitiendo una modificación posterior.

`hostcomplete`
: Si está habilitada, y se está usando Readline, Bash tratará de realizar una compleción del nombre de anfitrión cuando una palabra que contiene un '`@`' esté siendo completada (véase Sección 8.4.6 [Órdenes para Compleción], página 145). Esta opción está habilitada por defecto.

`huponexit`
: Si está establecida, Bash enviará `SIGHUP` a todas las tareas cuando se cierre un intérprete de acceso interactivo (véase Sección 3.7.6 [Señales], página 45).

`inherit_errexit`
: Si está establecida, la sustitución de instrucciones hereda el valor de la opción `errexit`, en vez de eliminarlo en el entorno del subintérprete. Esta opción es habilitada cuando se habilita el modo POSIX.

`interactive_comments`
: Permite que una palabra que comience por '`#`' haga que esa palabra y todos los caracteres restantes en esa línea sean ignorados en un intérprete interactivo. Esta opción está habilitada por defecto.

`lastpipe`
: Si está habilitada, y el control de tareas no está activado, el intérprete ejecuta la última instrucción de una tubería no ejecutada en segundo plano en el entorno actual del intérprete.

`lithist`
: Si está habilitada, y la opción `cmdhist` está habilitada, las instrucciones de múltiples líneas son guardadas en el historial con nuevas líneas incorporadas en vez de usar separadores de puntos y comas donde sea posible.

`localvar_inherit`
: Si está habilitada, las variables locales heredan el valor y los atributos de una variable del mismo nombre que exista en un alcance previo antes de que se asigne un nuevo valor. El atributo *nameref* no se hereda.

`localvar_unset`
: Si está habilitada, llamar a `unset` con variables locales en alcances previos de funciones las marca de forma que las siguientes consultas las encuentran sin asignar hasta que la función retorne. Esto es idéntico al comportamiento de eliminar variables locales en el alcance actual de la función.

`login_shell`
: El intérprete establece esta opción si se inicia como un intérprete de acceso (véase Sección 6.1 [Llamando a Bash], página 95). El valor no puede ser cambiado.

`mailwarn`
: Si está habilitada, y un archivo que Bash está comprobando en busca de correo ha sido accedido desde la última hora en que fue comprobado, el mensaje "`El correo en` *`mailfile`* `ha sido leído`" será mostrado.

`no_empty_cmd_completion`
: Si está habilitada, y se está usando Readline, Bash no tratará de buscar en el `PATH` para posibles compleciones cuando la compleción se intenta en una línea vacía.

`nocaseglob`
: Si está habilitada, Bash hace coincidir nombres de archivo independientemente de mayúsculas y minúscula al realizar la expansión de nombre de archivo.

`nocasematch`
: Si está habilitada, Bash hace coincidir patrones independientemente de mayúsculas y minúsculas al realizar coincidencias mientras ejecuta las instrucciones condicionales `case` o `[[`, cuando realiza expansiones de palabra de sustitución de patrón o cuando filtra posibles compleciones como parte de compleción programable.

`nullglob`
: Si está habilitada, Bash permite que patrones de nombre de archivo que no corresponden con ningún archivo se expandan a una cadena vacía, en vez de a sí mismos.

`progcomp`
: Si está habilitada, las facilidades programables de compleción (véase Sección 8.6 [Compleción Programable], página 150) son habilitadas. Está opción está habilitada por defecto.

`progcomp_alias`
: Si está habilitada y la compleción programable está activada, Bash trata un nombre de instrucción que no tiene ninguna compleción como un posible alias e intenta la expansión de alias. Si tiene un alias, Bash intenta la compleción programable usando la palabra de instrucción resultante del alias expandido.

`promptvars`
: Si está habilitada, las cadenas de prompt experimentan la expansión de parámetros, sustitución de instrucciones, expansión aritmética y eliminación de comillas después de ser expandidas como se describe más adelante (véase Sección 6.9 [Controlando el Prompt], página 109). Esta opción está habilitada por defecto.

`restricted_shell`
: El intérprete establece esta opción si es iniciado en modo restringido (véase Sección 6.10 [El Intérprete Restringido], página 110). El valor no puede ser cambiado. Esto no se elimina cuando los archivos de arranque se ejecutan, permitiendo a los archivos de arranque descubrir si el intérprete está restringido o no.

`shift_verbose`
: Si esto está establecido, la instrucción integrada `shift` imprime un mensaje de error cuando la cuenta de shift excede el número de parámetros posicionales.

`sourcepath`
: Si está habilitada, la instrucción integrada `source` usa el valor de `PATH` para encontrar el directorio que contiene el archivo proporcionado como un argumento. Está opción está habilitada por defecto.

`xpg_echo`
: Si está habilitada, la instrucción integrada `echo` expande por defecto las secuencias de escape de barras invertidas.

4.4 Instrucciones Integradas Especiales

Por razones históricas, el estándar POSIX ha clasificado varias instrucciones integradas como *especiales*. Cuando Bash se ejecuta en modo POSIX, las instrucciones integradas especiales difieren de otras instrucciones integradas en tres aspectos:

1. Las instrucciones integradas especiales son encontradas antes que las funciones del intérprete durante la búsqueda de órdenes.
2. Si una instrucción integrada especial devuelve un estado de error, un intérprete no interactivo se cierra.
3. Las sentencias de asignación que preceden a la instrucción se mantienen en efecto en el entorno del intérprete después de que la instrucción se complete.

Cuando Bash no se ejecuta en modo POSIX, estas instrucciones integradas no se comportan de modo diferente al del resto de instrucciones integradas de Bash. El modo POSIX de Bash se describe en Sección 6.11 [Modo POSIX de Bash], página 111.

Estas son las instrucciones integradas especiales POSIX:

```
break : . continue eval exec exit export readonly return set
shift trap unset
```

5 Variables del Intérprete

Este capítulo describe las variables del intérprete que usa Bash. Bash asigna automáticamente valores predeterminados a varias variables.

5.1 Variables del Bourne Shell

Bash usa ciertas variables del intérprete de la misma forma que el Bourne shell. En algunos casos, Bash asigna un valor predeterminado a la variable.

`CDPATH` Una lista de directorios separada por dos puntos usada como una ruta de búsqueda para la instrucción integrada `cd`.

`HOME` El directorio personal del usuario actual; el predeterminado para la instrucción integrada `cd`. El valor de esta variable es usado también por la expansión de virgulilla (véase Sección 3.5.2 [Expansión de Virgulilla], página 26).

`IFS` Una lista de caracteres que separa campos; usada cuando el intérprete divide palabras como partes de expansión.

`MAIL` Si este parámetro está establecido a un nombre de archivo o directorio y la variable `MAILPATH` no está asignada, Bash informa al usuario de la llegada de correo en el archivo especificado o directorio con formato de Maildir.

`MAILPATH` Una lista separada por dos puntos de nombres de archivos que el intérprete comprueba periódicamente en busca de nuevo correo. Cada entrada de la lista puede especificar el mensaje que es imprimido cuando llega correo nuevo en el archivo de correo separando el nombre de archivo del mensaje con un '`?`'. Cuando se usa en el texto del mensaje, `$_` se expande al nombre del actual archivo de correo.

`OPTARG` El valor del último argumento de opción procesado por la instrucción integrada `getopts`.

`OPTIND` El índice del último argumento de opción procesado por la instrucción integrada `getopts`.

`PATH` Una lista de directorios separada por dos puntos en la que el intérprete busca instrucciones. Un nombre de directorio de longitud cero (nulo) en el valor de `PATH` indica el directorio actual. Un nombre de directorio nulo puede aparecer como dos dos puntos adyacentes o como dos puntos al final o al principio.

`PS1` La cadena de prompt primaria. El valor predeterminado es '`\s-\v\$ `'. Véase Sección 6.9 [Controlando el Prompt], página 109, para la lista completa de secuencias de escape que son expandidas antes de que se muestre `PS1`.

`PS2` La cadena de prompt secundaria. El valor predeterminado es '`>`'. `PS2` se expande de la misma forma que `PS1` antes de ser mostrado.

5.2 Variables de Bash

Estas palabras son establecidas o usadas por Bash, pero otros intérpretes no las tratan normalmente de forma especial.

Unas pocas variables usadas por Bash son descritas en diferentes capítulos: variables para controlar las herramientas de control de tareas (véase Sección 7.3 [Variables de Control de Tareas], página 121).

`_` ($_, una barra baja.) Durante el inicio del intérprete, se establece al nombre de ruta usado para llamar al intérprete o al guion del intérprete ejecutado como pasado en el entorno o la lista de argumentos. Posteriormente expande al último argumento a la anterior instrucción simple ejecutada en primer plano, después de la expansión. También establece el nombre de ruta completa para llamar a cada instrucción ejecutada y ubicada en el entorno exportado para esa instrucción. Al comprobar el correo, este parámetro mantiene el nombre del archivo de correo.

`BASH` El nombre de ruta completo usado para ejecutar la actual instancia de Bash.

`BASHOPTS` Una lista separada por dos puntos de opciones del intérprete habilitadas. Cada palabra en la lista es un argumento válido para la opción `-s` de la instrucción integrada `shopt` (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73). Las opciones que aparecen en `BASHOPTS` son aquellas que se muestran como '`on`' por '`shopt`'. Si esta variable está en el entorno cuando Bash se inicia, cada opción del intérprete en la lista será activada antes de leer cualquier archivo de inicio. La variable es de solo lectura.

`BASHPID` Se expande al ID de proceso del proceso actual de Bash. Esto difiere de `$$` en ciertas circunstancias, como con subintérpretes que no necesitan de Bash para ser reinicializados. Las asignaciones a `BASHPID` no tienen efecto. Si `BASHPID` no está asignada, pierde sus propiedades especiales, incluso si se restablece posteriormente.

`BASH_ALIASES`
Una variable de vector asociativo cuyos miembros corresponden a la lista interna de aliases según mantiene la instrucción integrada `alias` (véase ⟨undefined⟩ [Instrucciones Integradas del Intérprete Bourne], página ⟨undefined⟩). Los elementos añadidos a este vector aparecen en la lista de aliases; sin embargo, desasignar los elementos del vector actualmente no hace que los aliases sean eliminados de la lista de aliases. Si `BASH_ALIASES` está sin asignar, pierde sus propiedades especiales, incluso si es restablecida posteriormente.

`BASH_ARGC`
Una variable de vector cuyos valores son el número de parámetros en cada marco de la actual pila de llamadas de ejecución de bash. El número de parámetros para la subrutina actual (función del intérprete o guion ejecutado con `.` o `source`) está encima en la pila. Cuando se ejecuta una subrutina, el número de parámetros pasados se mete en `BASH_ARGC`. El intérprete asigna `BASH_ARGC` solo cuando está en el modo de depuración extendido (consulte Sección 4.3.2 [La Instrucción Integrada Shopt], página 73, para una descripción de la opción `extdebug` para la instrucción integrada `shopt`). Asignar `extdebug` después de

que se haya iniciado el intérprete para ejecutar un guion o referenciar esta variable cuando `extdebug` no está activada, puede producir valores inconsistentes.

`BASH_ARGV`
: Una variable de vector que contiene todos los parámetros en la pila de ejecución actual de bash. El parámetro final de la última llamada de subrutina está encima en la pila; el primer parámetro de la llamada inicial está abajo. Cuando se ejecuta una subrutina, los parámetros proporcionados se meten en `BASH_ARGV`. El intérprete asigna `BASH_ARGV` solo cuando está en el modo de depuración extendido (vea Sección 4.3.2 [La Instrucción Integrada Shopt], página 73, para una descripción de la opción `extdebug` de la instrucción integrada `shopt`). Establecer `extdebug` después de que el intérprete ha empezado a ejecutar un guion o hacer referencia a esta variable cuando `extdebug` no está habilitado, puede producir valores inconsistentes.

`BASH_ARGV0`
: Cuando se le hace referencia, esta variable se expande al nombre del intérprete o guion del intérprete (idéntica a `$0`; Véase Sección 3.4.2 [Parámetros Especiales], página 23, para una descripción del parámetro especial 0). La asignación a `BASH_ARGV0` hace que el valor asignado también sea asignado a `$0`. Si `BASH_ARGV0` no está asignado, pierde sus propiedades especiales, incluso si incluso si es restablecida posteriormente.

`BASH_CMDS`
: Una variable de vector asociativo cuyos miembros corresponden a la tabla hash interna de instrucciones según la mantiene la instrucción integrada `hash` (véase ⟨undefined⟩ [Instrucciones Integradas del Intérprete Bourne], página ⟨undefined⟩). Los elementos añadidos a este vector aparecen en la tabla hash; sin embargo, desasignar elementos del vector no hace que los nombres de las instrucciones sean eliminados de la tabla hash. Si `BASH_CMDS` no está asignada, pierde sus propiedades especiales, incluso si incluso si es restablecida posteriormente.

`BASH_COMMAND`
: La instrucción actualmente en ejecución o a punto de ser ejecutada, a no ser que el intérprete está ejecutando una instrucción como el resultado de una trampa, en cuyo caso, es la instrucción que se ejecuta sobre la trampa. Si `BASH_COMMAND` no está asignada, pierde sus propiedades especiales, incluso si incluso si es restablecida posteriormente.

`BASH_COMPAT`
: El valor se usa para establecer el nivel de compatibilidad del intérprete. Véase Sección 6.12 [Modo de Compatibilidad del Intérprete], página 115, para una descripción de los diversos niveles de compatibilidad y sus efectos. El valor puede ser un número decimal (p. ej., 4,2) o un entero (p. ej., 42) que corresponde al nivel de compatibilidad deseado. Si la variable `BASH_COMPAT` no está asignada o está asignada a una cadena vacía, el nivel de compatibilidad se establece al predeterminado para la versión actual. Si a `BASH_COMPAT` se le asigna un valor que no uno de los niveles válidos de compatibilidad, el intérprete muestra un mensaje de error y establece el nivel de compatibilidad al predeterminado para

la versión actual. Los valores válidos corresponden a los niveles de compatibilidad descritos a continuación (véase Sección 6.12 [Modo de Compatibilidad del Intérprete], página 115). Por ejemplo, 4.2 y 42 son valores válidos que corresponden a la opción `shopt compat42` y establecen el nivel de compatibilidad en 42. La versión actual también es un valor válido.

`BASH_ENV` Si esta variable está asignada cuando Bash es llamado para ejecutar un guion del intérprete, su valor se expande y se usa como el nombre de un archivo de inicio que leer antes de ejecutar el guion. Véase Sección 6.2 [Archivos de Inicio de Bash], página 97.

`BASH_EXECUTION_STRING`
El argumento de instrucción para la opción de llamada `-c`.

`BASH_LINENO`
Una variable de vector cuyos miembros son los números de línea en archivos fuentes donde se invocó cada miembro correspondiente de *FUNCNAME*. `${BASH_LINENO[$i]}` es el número de línea en el archivo fuente (`${BASH_SOURCE[$i+1]}`) donde `${FUNCNAME[$i]}` fue llamado (o `${BASH_LINENO[$i-1]}` si se referencia dentro de otra función del intérprete). Use `LINENO` para obtener el número de línea actual.

`BASH_LOADABLES_PATH`
Una lista de directorios separada por dos puntos en la que el intérprete busca instrucciones integradas dinámicamente cargables especificadas por la instrucción `enable`.

`BASH_REMATCH`
Un vector cuyos miembros están asignados por el operador binario '`=~`' a la instrucción condicional (véase Sección 3.2.5.2 [Construcciones Condicionales], página 12). El elemento de índice 0 es la porción de la cadena que coincide con la expresión regular entera. El elemento de índice *n* es la porción de la cadena que coincide con la subexpresión número *n* entre paréntesis.

`BASH_SOURCE`
Una variable de vector cuyos miembros son los nombres de archivo fuentes donde están definidos los correspondientes nombres de funciones del intérprete en la variable de vector `FUNCNAME`. La función del intérprete `${FUNCNAME[$i]}` está definida en el archivo `${BASH_SOURCE[$i]}` y es llamada desde `${BASH_SOURCE[$i+1]}`

`BASH_SUBSHELL`
Se incrementa en uno dentro de cada subintérprete o entorno de subintérprete cuando el intérprete empieza a ejecutarse en ese entorno. El valor inicial es 0. Si `BASH_SUBSHELL` no está asignada, pierde sus propiedades especiales, incluso si incluso si es restablecida posteriormente.

`BASH_VERSINFO`
Una variable de vector de solo lectura (véase Sección 6.7 [Vectores], página 105) cuyos miembros guardan información de versión para esta instancia de Bash. Los valores asignados a los miembros del vector son los siguientes:

`BASH_VERSINFO[0]`
: El número de versión mayor (la *publicación*).

`BASH_VERSINFO[1]`
: El número de versión menor (la *versión*).

`BASH_VERSINFO[2]`
: El nivel de parche.

`BASH_VERSINFO[3]`
: La versión de construcción.

`BASH_VERSINFO[4]`
: El estado de publicación (p. ej., *beta1*).

`BASH_VERSINFO[5]`
: El valor de `MACHTYPE`.

`BASH_VERSION`
: El número de versión de la actual instancia de Bash.

`BASH_XTRACEFD`
: Si está asignado a un entero correspondiente a un descriptor de archivo válido, Bash escribirá el rastreo de salida generado cuando la '`set -x`' está habilitado para ese descriptor de archivo. Esto permite que el rastreo de salida esté separado de mensajes de diagnóstico y error. El descriptor de archivo se cierra cuando se elimina o se asigna un nuevo valor a `BASH_XTRACEFD`. Eliminar `BASH_XTRACEFD` o asignarlo a la cadena vacía hace que el rastreo de salida sea asignado al error estándar. Observe que asignar `BASH_XTRACEFD` a 2 (el descriptor de archivo del error estándar) y después eliminarlo hará que sea cerrado el error estándar.

`CHILD_MAX`
: Establece el número de valores de estados de salida finalizados que el intérprete puede recordar. Bash no permitirá que este valor sea reducido por debajo de un mínimo mandado por POSIX, y hay un valor máximo (actualmente 8192) que no puede ser excedido. El valor mínimo es dependiente del sistema.

`COLUMNS`
: Usada por la instrucción `select` para determinar la anchura de la terminal al imprimir listas de selección. Automáticamente asignada si la opción `checkwinsize` está activada (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73), o en un intérprete interactivo al recibir una `SIGWINCH`.

`COMP_CWORD`
: Un indice a `${COMP_WORDS}` de la palabra que contiene la posición actual del cursor. Esta variable está disponible solo en funciones del intérprete llamadas por las herramientas de compleción programables (véase Sección 8.6 [Compleción Programable], página 150).

`COMP_LINE`
: La línea de orden actual. Esta variable está disponible solo en funciones del intérprete e instrucciones externas llamadas por las herramientas de compleción programables (véase Sección 8.6 [Compleción Programable], página 150).

`COMP_POINT`
: El índice de la posición actual del cursor relativa al comienzo de la instrucción actual. Si la posición actual del cursor está al final de la instrucción actual, el valor de esta variable es igual a `${#COMP_LINE}`. Esta variable está disponible solo en funciones del intérprete e instrucciones externas llamadas por las herramientas de compleción programables (véase Sección 8.6 [Compleción Programable], página 150).

`COMP_TYPE`
: Asignada a un valor entero correspondiente al tipo de compleción intentado que hizo que una función de compleción fuera llamada: *TAB*, para compleción normal; '`?`', para listar compleciones después de sucesivas tabulaciones, '`!`', para listar alternativas a compleción parcial de palabra, '`@`', para listar compleciones si la palabra no se modifica; o '`%`', para compleción de menú. Esta variable está disponible solo en funciones del intérprete e instrucciones externas llamadas por las herramientas de compleción programables (véase Sección 8.6 [Compleción Programable], página 150).

`COMP_KEY`
: La tecla (o tecla final de una secuencia de teclas) usada para llamar a la actual función de compleción.

`COMP_WORDBREAKS`
: El conjunto de caracteres que la librería Readline trata como separadores de palabras al realizar la compleción de palabras. Si `COMP_WORDBREAKS` está sin asignar, pierde sus propiedades especiales, incluso si incluso si es restablecida posteriormente.

`COMP_WORDS`
: Una variable de vector que consta de las palabras individuales en la actual línea de órdenes. La línea se divide en palabras como Readline la dividiría, usando `COMP_WORDBREAKS` como se describe anteriormente. Esta variable está disponible solo en funciones del intérprete llamadas por las herramientas de compleción programables (véase Sección 8.6 [Compleción Programable], página 150).

`COMPREPLY`
: Una variable de vector de la que Bash lee las posibles compleciones generadas por una función del intérprete llamada por la herramienta de compleción programable (véase Sección 8.6 [Compleción Programable], página 150). Cada elemento del vector contiene una posible compleción.

`COPROC`
: Una variable de vector creada para alojar los descriptores de archivo para salida de y entrada para un coproceso sin nombre (véase Sección 3.2.6 [Coprocesos], página 17).

`DIRSTACK`
: Una variable de vector que contiene los contenidos actuales de la pila de directorios. Los directorios aparecen en la pila en el orden en que son mostrados por la instrucción integrada `dirs`. Asignar a miembros de esta variable de vector puede usarse para modificar directorios ya en la pila, pero las instrucciones integradas `pushd` y `popd` deben ser usadas para añadir y eliminar directorios. La asignación a esta variable no cambiará el directorio actual. Si `DIRSTACK` está sin asignar, pierde sus propiedades especiales, incluso si es restablecida posteriormente.

`EMACS` Si Bash encuentra esta variable en el entorno cuando el intérprete se inicia con el valor '`t`', asume que el intérprete está corriendo en un búfer de intérprete de Emacs y deshabilita la edición de línea.

`ENV` Expandido y ejecutado de forma similar a `BASH_ENV` (véase ⟨undefined⟩ [Archivos de Inicio de Bahs], página ⟨undefined⟩) cuando se llama a un intérprete interactivo en el modo POSIX (véase Sección 6.11 [Modo POSIX de Bash], página 111).

`EPOCHREALTIME`
Cada vez que se hace referencia a este parámetro se expande al número de segundos desde la Época Unix como un valor de coma flotante con granularidad de microsegundos (consulte la documentación de la función de la librería C *time* para la definición de Época). Las asignaciones a `EPOCHREALTIME` son ignoradas. Si `EPOCHREALTIME` es eliminada, pierde sus propiedades espciales, incluso si es restablecida posteriormente.

`EPOCHSECONDS`
Cada vez que se hace referencia a este parámetro, se expande al número de segundos desde el Tiempo Unix (consulte la definición de Época en la documentación de la función *time* de la librería C). Las asignaciones a `EPOCHSECONDS` son ignoradas. Si `EPOCHSECONDS` no está asignada, pierde sus propiedades especiales, incluso si es restablecida posteriormente.

`EUID` El id numérico efectivo de usuario del actual usuario. Esta variable es de solo lectura.

`EXECIGNORE`
Una lista de patrones separada por dos puntos (véase Sección 3.5.8.1 [Coincidencia de Patrones], página 36) que define la lista de nombres de archivo que serán ignorados por la búsqueda de instrucciones usando `PATH`. Los archivos cuyos nombres de ruta completos coinciden con uno de estos patrones no se consideran archivos ejecutables para los propósitos de compleción y ejecución mediante búsqueda de *PATH*. Esto no afecta el comportamiento de las instrucciones `[`, `test` y `[[`. Los nombres de ruta completos en la tabla hash de instrucciones no están sujetos a `EXECIGNORE`. Use esta variable para ignorar los archivos de librería compartidos que tiene asignado el bit de ejecución, pero no son archivos ejecutables. La coincidencia de patrones hace honor al ajuste de la opción del intérprete `extglob`.

`FCEDIT` El editor usado como predeterminado por la opción `-e` de la instrucción integrada `fc`.

`FIGNORE` Una lista separada de sufijos que ignorar al realizar la compleción de nombre de archivo. Un nombre de archivo cuyo sufijo coincide con una de las entradas en `FIGIGNORE` es excluido de la lista de nombres de archivo completados. Un valor de muestra es '`.o:~`'.

`FUNCNAME` Una variable de vector que contiene los nombres de todas las funciones del intérprete actualmente en la pila de ejecución de llamadas. El elemento con índice 0 es el nombre de cualquier función del intérprete en ejecución actualmente. El elemento más al fondo (aquel con el índice más alto) es `"main"`. Esta

variable solo existe cuando una función del intérprete se está ejecutando. Las asignaciones a `FUNCNAME` no tienen efecto. Si `FUNCNAME` está sin asignar, pierde sus propiedades especiales, incluso si es restablecida posteriormente.

Esta variable puede ser usada con `BASH_LINENO` y `BASH_SOURCE`. Cada elemento de `FUNCNAME` tiene elementos correspondientes en `BASH_LINENO` y `BASH_SOURCE` para describir la pila de llamadas. Por ejemplo, `${FUNCNAME[$i]}` fue llamada del archivo `${BASH_SOURCE[$i+1]}` en el número de línea `${BASH_LINENO[$i]}`. La instrucción integrada `caller` muestra la actual pila de llamadas usando esta información.

`FUNCNEST` Si está asignada a un valor numérico mayor que 0, define un nivel de anidamiento de función máximo. Las llamadas de función que excedan este límite harán que se aborte la instrucción actual.

`GLOBIGNORE`
Una lista de patrones separada por dos puntos que define el número de nombres de archivo que deben ser ignorados por la expansión de nombre de archivo. Si un nombre de archivo coincidido por un patrón de expansión de nombre de archivo también coincide con uno de los patrones en `GLOBIGNORE`, es eliminado de la lista de coincidencias. Los patrones coincidentes hacen honor al ajuste de la opción del intérprete `extglob`.

`GROUPS` Una variable de vector que contiene la lista de grupos de los cuales el usuario actual es miembro. Las asignaciones a `GROUPS` no tienen efecto. Si `GROUPS` es eliminada, pierde sus propiedades especiales, incluso si es restablecida posteriormente.

`histchars`
Hasta tres caracteres que controlan la expansión de historial, sustitución rápida y simbolización (véase Sección 9.3 [Interacción con el Historial], página 163). El primer carácter es el carácter de *expansión de historial*, es decir, el carácter que representa el inicio de una expansión de historial, normalmente '`!`'. El segundo carácter es el carácter que representa 'sustitución rápida cuando es visto como el primer carácter en la línea, normalmente '`^`'. El tercer carácter opcional es el carácter que indica que el resto de la línea es un comentario cuando se encuentra como el primer carácter de una palabra, normalmente '`#`'. El carácter de comentario de historial hace que sea saltada la sustitución de historial para el resto de palabras en la línea. No necesariamente hace que el analizador del intérprete trate el resto de la línea como un comentario.

`HISTCMD` El número de historial, o el índice en la lista del historial, de la instrucción actual. Las asignaciones a `HISTCMD` son ignoradas. Si `HISTCMD` no está asignada, pierde sus propiedades especiales, incluso si es restablecida posteriormente.

`HISTCONTROL`
Una lista de valores separada por dos puntos que controla cómo se guardan las instrucciones en la lista de historial. Si la lista de valores incluye '`ignorespace`', las líneas que empiezan por un carácter de espacio no se guardan en la lista del historial. Un valor de '`ignoredups`' hace que las líneas que coinciden con la entrada de historial anterior no sean guardadas. Un valor de '`ignoreboth`' es una

abreviatura de '`ignorespace`' e '`ignoredups`'. Un valor de '`erasedups`' hace que todas las líneas anteriores que coinciden con la línea actual sean eliminadas de la lista del historial antes de que la línea sea guardada. Cualquier valor fuera de la lista anterior se ignora. Si `HISTCONTROL` está sin asignar o no incluye un valor válido, todas las líneas leídas por el analizador del intérprete son guardadas en la lista del historial, sujetas al valor de `HISTIGNORE`. La segunda y posteriores líneas de una instrucción multilínea compuesta no son probadas y se añaden al historial sin importar el valor de `HISTCONTROL`.

`HISTFILE` El nombre del archivo al cual se guarda el historial de instrucciones. El valor predeterminado es `~/.bash_history`.

`HISTFILESIZE`
El máximo número de líneas contenidas en el archivo de historial. Cuando se asigna a esta variable un valor, el archivo de historial es recortado, si es necesario, para contener no más del número de líneas eliminando las entradas más antiguas. El archivo de historial también es recortado a este tamaño después de escribirlo cuando finaliza un intérprete. Si el valor es 0, el archivo de historial se recorta a tamaño cero. Las valores no numéricos y valores numéricos menores de cero inhiben el recorte. El intérprete asigna el valor predeterminado al valor de `HISTSIZE` después de leer los archivos de inicialización.

`HISTIGNORE`
Una lista de patrones separada por dos puntos usada para decidir qué líneas de instrucción deberían guardarse en la lista del historial. Cada patrón es anclado al principio de la línea y debe coincidir con la línea completa (no se añade un '`*`' implícito). Cada patrón se comprueba contra la línea después de que las comprobaciones especificadas por `HISTCONTROL` sean aplicadas. Además de los caracteres normales de coincidencia de patrones del intérprete, '`&`' coincide con la anterior línea de historial. '`&`' puede ser escapada usando una barra invertida; la barra invertida se elimina antes de intentar una coincidencia. La segunda y las posteriores líneas de una instrucción compuesta multilínea no son comprobadas, y se añaden al historial sin importar el valor de `HISTIGNORE`. El patrón que coincide hace honor al ajuste de la opción del intérprete `extglob`.

`HISTIGNORE` subsume la función de `HISTCONTROL`. Un patrón de '`&`' es idéntico a `ignoredups`, y un patrón de '`[ ]*`' es idéntico a `ignorespace`. Combinar estos dos patrones, separándolos con dos puntos, proporciona la funcionalidad de `ignoreboth`.

`HISTSIZE` El máximo número de instrucciones que recordar en la lista del historial. Si el valor es 0, las instrucciones no se guardan en la lista del historial. Los valores numéricos menores que cero resultan en cada instrucción guardándose en la lista del historial (no hay límite). El intérprete establece el valor predeterminado a 500 después de leer los archivos de inicio.

`HISTTIMEFORMAT`
Si esta variable está asignada y no es nula, su valor se usa como una cadena de formato para *strftime* para imprimir la marca temporal asociada con cada entrada de historial mostrada por la instrucción integrada `history`. Si está asignada esta variable, las marcas de tiempo se escriben al archivo de historial

de forma que puedan ser preservadas entre sesiones del intérprete. Esto usa el carácter de comentario de historial para distinguir marcas de tiempo de otras líneas de historial.

`HOSTFILE` Contiene el nombre de un archivo en el mismo formato que `/etc/hosts` que debería ser leído cuando el intérprete necesita completar un nombre de anfitrión. La lista de posibles compleciones de nombre de anfitrión puede ser cambiada mientra el intérprete se ejecuta; la siguiente vez que se intente la compleción de nombre de anfitrión después de que se cambie el valor, Bash añade los contenidos del nuevo archivo a la lista existente. Si `HOSTFILE` está asignada pero no tiene valor o no nombra un archivo legible, Bash trata de leer `/etc/hosts` para obtener la lista de posibles compleciones de nombre de anfitrión. Cuando `HOSTFILE` está sin asignar, se limpia la lista de nombre de anfitrión.

`HOSTNAME` El nombre del anfitrión actual.

`HOSTTYPE` Una cadena que describe la máquina en que se está ejecutando Bash.

`IGNOREEOF`
Controla la acción del intérprete al recibir un carácter `EOF` como la única entrada. Si está asignado, el valor denota el número de caracteres `EOF` consecutivos que pueden ser leídos como el primer carácter de una línea de entrada antes de que finalice el intérprete. Si la variable existe pero no tiene un valor numérico, o no tiene valor, entonces el predeterminado es 10. Si la variable no existe, el `EOF` significa el final de la entrada para el intérprete. Esto solo está en efecto para intérpretes interactivos.

`INPUTRC` El nombre del archivo de inicialización Readline, que sobrescribe el valor predeterminado de `~/.inputrc`.

`INSIDE_EMACS`
Si Bash encuentra esta variable en el entorno cuando el intérprete se inicia, asume que el intérprete está corriendo en un búfer de intérprete de Emacs y puede deshabilitar la edición de línea dependiendo del valor de `TERM`.

`LANG` Usada para determinar la categoría de configuración regional para cualquier categoría no especifícamente seleccionada con una variable que empieza por `LC_`.

`LC_ALL` Esta variable sobrescribe el valor de `LANG` y cualquier otra variable `LC_` que especifica una categoría de configuración regional.

`LC_COLLATE`
Esta variable determina la ordenación usada para ordenar los resultados de la expansión de nombre de archivo, y determina el comportamiento de las expresiones de rango, clases de equivalencia y secuencias de ordenación dentro de la expansión de nombre de archivo y coincidencia de patrones (véase Sección 3.5.8 [Expansión de Nombre de Archivo], página 35).

`LC_CTYPE` Esta variable determina la interpretación de caracteres y el comportamiento de las clases de caracteres dentro de una expansión de nombre de archivo y coincidencia de patrones (véase Sección 3.5.8 [Expansión de Nombre de Archivo], página 35).

`LC_MESSAGES`
: Esta variable determina la configuración regional usada para traducir cadenas entre comillas dobles precedidas por un '`$`' (véase Sección 3.1.2.5 [Traducción de Localización], página 7).

`LC_NUMERIC`
: Esta variable determina la categoría de configuración regional para el formato de números.

`LC_TIME`
: Esta variable determina la categoría de configuración regional usada para el formato de fecha y tiempo.

`LINENO`
: El número de línea en el guion o en la función actualmente en ejecución. Si `LINENO` no está asignada, pierde sus propiedades especiales, incluso si es restablecida posteriormente.

`LINES`
: Usada por la instrucción `select` para determinar la longitud de columna para imprimir listas de selección. Automáticamente asignada si la opción `checkwinsize` está activada (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73), o en un intérprete interactivo al recibir una `SIGWINCH`.

`MACHTYPE`
: Una cadena que describe completamente el tipo de sistema en que Bash está ejecutándose, en el formato estándar GNU *ucp-empresa-sistema*.

`MAILCHECK`
: La frecuencia (en segundos) con la que el intérprete debe comprobar el correo en los archivos especificados en las variables `MAILPATH` o `MAIL`. La predeterminada es 60 segundos. Cuando es hora de comprobar el correo, el intérprete lo hace antes de mostrar el prompt primario. Si esta variable está sin asignar o asignada a un valor que no es un número mayor o igual a cero, el intérprete deshabilita la comprobación de correo.

`MAPFILE`
: Una variable de vector creada para contener el texto leído por la instrucción integrada `mapfile` cuando no se proporciona nombre de variable.

`OLDPWD`
: El directorio de trabajo anterior según se establece por la instrucción integrada `cd`.

`OPTERR`
: Si asignada al valor 1, Bash muestra mensajes de error generados por la instrucción integrada `getopts`.

`OSTYPE`
: Una cadena que describe el sistema operativo en que Bash se está ejecutando.

`PIPESTATUS`
: Una variable de vector (véase Sección 6.7 [Vectores], página 105) que contiene una lista de los valores de estado de salida de los procesos en la tubería en primer plano más recientemente ejecutada (que podría contener una sola instrucción).

`POSIXLY_CORRECT`
: Si esta variable está en el entorno cuando Bash se inicia, el intérprete entra al modo POSIX (véase Sección 6.11 [Modo POSIX de Bash], página 111) antes de leer los archivos de inicio, como si se hubiera proporcionado la opción de

llamada `--posix`. Si se asigna cuando el intérprete está en ejecución, Bash activa el modo POSIX, como si la instrucción

```
set -o posix
```

hubiera sido ejecutada. Cuando el intérprete entra en modo POSIX asigna la variable si no a sido aún asignada.

`PPID` El ID de proceso del proceso padre del intérprete. Esta variable es de solo lectura.

`PROMPT_COMMAND`
Si esta variable está asignada y es un vector, el valor de cada elemento asignado se interpreta como una instrucción a ejecutar antes de mostrar el prompt primario (`$PS1`). Si esto está asignado pero no una variable de vector, su valor se usa en su lugar como una instrucción a ejecutar.

`PROMPT_DIRTRIM`
Si está asignada a un número mayor que cero, el valor se usa como el número de componentes finales de directorio que retener al expandir las cadenas de escape de prompt `\w` y `\W` (véase Sección 6.9 [Controlando el Prompt], página 109). Los caracteres eliminados son reemplazados por puntos suspensivos.

`PS0` El valor de este parámetro se expande como `PS1` y se muestra mediante intérpretes interactivos después de leer una instrucción y antes de que se ejecute la instrucción.

`PS3` El valor de esta variable se usa como el prompt para la instrucción `select`. Si no está asignada esta variable, la instrucción `select` usa el prompt '`#?` '.

`PS4` El valor de este parámetro se expande como *PS1* y el valor expandido es el prompt imprimido antes de que la línea de instrucción sea repetida cuando la opción `-x` esté habilitada (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68). El primer carácter del valor expandido se repete múltiples veces, según sea necesario, para indicar múltiples niveles de indirección. El predeterminado es '`+`'.

`PWD` El directorio de trabajo actual según se establece por la instrucción integrada `cd`.

`RANDOM` Cada vez que se hace referencia a este parámetro, se expande a un entero aleatorio entre 0 y 32767. Asignar un valor a esta variable crea una semilla para el generador de números aleatorios. Si `RANDOM` está sin asignar, pierde sus propiedades especiales, incluso si se restablece posteriormente.

`READLINE_LINE`
Los contenidos del búfer de línea de Readline, para uso con '`bind -x`' (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).

`READLINE_MARK`
La posición de la *marca* (punto de inserción guardado) en el búfer de línea de Readline, para el uso con '`bind -x`' (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56). Los caracteres entre el punto de inserción y la marca son llamados normalmente *región*.

`READLINE_POINT`
: La posición del punto de inserción en el búfer de línea de Readline, para uso con '`bind -x`' (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).

`REPLY` La variable predeterminada para la instrucción integrada `read`.

`SECONDS` Esta variable se expande al número de segundos desde que el intérprete se inició. La asignación a esta variable restablece el contador para el valor asignado, y el valor expandido se convierte en el valor asignado más el número de segundos desde la asignación. El número de segundos en la llamada al intérprete y el tiempo actual siempre se determinan consultando el reloj del sistema. Si `SECONDS` no está asignada, pierde sus propiedades especiales, incluso si se restablece posteriormente.

`SHELL` La variable de entorno se expande al nombre de ruta completo del intérprete. Si no está asignada cuando el intérprete se inicia, Bash la asigna al nombre de ruta compreto del intérprete de inicio de sesión del usuario actual.

`SHELLOPTS`
: Una lista de opciones del intérprete habilitadas separada por dos puntos. Cada palabra en la lista es un argumento válido para la opción `-o` de la instrucción integrada `set` (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68). Las opciones que aparecen en `SHELLOPTS` son aquellas descritas con '`on`' por '`set -o`'. Si esta variable está en el entorno cuando Bash se inicia, cada opción del intérprete en la lista será activada antes de leer los archivos de inicio. Esta variable es de solo lectura.

`SHLVL` Incrementada en uno cada vez que se inicia una nueva instancia de Bash. Esto está pensado para ser un contador de la profundidad en que sus intérpretes de Bash están anidados.

`SRANDOM` Está variable se expande a un número pseudoaleatorio de 32 bits cada vez que se le hace referencia. El generador de números aleatorios no es linear en sistemas que soportan `/dev/urandom` o `arc4random`, de forma que cada número devuelto no tiene relación con los números que lo preceden. No se puede usar una semilla para el generador de números aleatorios, así que las asignaciones a esta variable no tienen efecto. Si `SRANDOM` no está asignada, pierde sus propiedades especiales, incluso si es posteriormente restablecida.

`TIMEFORMAT`
: El valor de este parámetro se usa como una cadena de formato que especifica cómo debería mostrarse la información temporal para tuberías prefijadas con la palabra reservada `time`. El carácter '`%`' introduce una secuencia de escape que se expande a un valor temporal u otra información. Las secuencias de escape y sus significados son los siguientes; las llaves denotan partes opcionales.

  `%%` Un '`%`' literal.

  `%[`*p*`][l]R` El tiempo transcurrido en segundos.

  `%[`*p*`][l]U` El número de segundos de UCP pasados en modo usuario.

  `%[`*p*`][l]S` El número de segundos de UCP pasados en modo sistema.

`%P` El porcentaje de UCP, computado como (%U + %S) / %R.

La *p* opcional es un dígito que especifica la precisión, el número de dígitos fraccionales después de la coma decimal. Un valor de 0 hace que no se muestre ninguna coma decimal o fracción. Como mucho pueden especificarse tres lugares tras la coma decimal; valores de *p* mayores que 3 son cambiados a 3. Si no se especifica *p*, se usa el valor 3.

El `l` opcional especifica un formato más largo, incluyendo minutos, de la forma *MM*m*SS.FF*s. El valor de *p* determina si la fracción es incluida o no.

Si esta variable no está asignada, Bash actúa como si tuviera el valor

```
$'\nreal\t%3lR\nuser\t%3lU\nsys\t%3lS'
```

Si el valor es nulo, no se muestra información de tiempo. Se añade una nueva línea final cuando se muestra la cadena de formato.

`TMOUT` Si está asignada a un valor mayor que cero, `TMOUT` se trata como el tiempo límite predeterminado para la instrucción integrada `read` (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56). La instrucción `select` (véase Sección 3.2.5.2 [Construcciones Condicionales], página 12) termina si la entrada no llega después de `TMOUT` segundos cuando la entrada viene de una terminal.

En un intérprete interactivo, el valor se interpreta como el número de segundos que esperar a una línea de entrada después de generar el prompt primario. Bash termina después de esperar ese número de segundos si no llega una línea de entrada completa.

`TMPDIR` Si está asignada, Bash usa su valor como el nombre de un directorio en el que Bash crea archivos temporales para que use el intérprete.

`UID` El id numérico real de usuario del usuario actual. Esta variable es de solo lectura.

6 Funcionalidades de Bash

Este capítulo describe funcionalidades únicas de Bash.

6.1 Llamando a Bash

```
bash [opción-larga] [-ir] [-abefhkmnptuvxdBCDHP] [-o opción]
     [-O opción_shopt] [argumento ...]
bash [opción-larga] [-abefhkmnptuvxdBCDHP] [-o opción]
     [-O opción_shopt] -c string [argumento ...]
bash [opción-larga] -s [-abefhkmnptuvxdBCDHP] [-o opción]
     [-O opción_shopt] [argumento ...]
```

Todas las opciones de un carácter usadas por la instrucción integrada `set` (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68) pueden ser usadas como opciones cuando se llama al intérprete. Además, hay varias opciones de múltiples caracteres que puede usar. Estas opciones tienen que aparecer en la línea de órdenes antes de las opciones de un carácter para ser reconocidas.

`--debugger`
: Se encarga de que el perfil del depurador se ejecute antes de que se inicie el intérprete. Activa el modo de depuración extendido (vea Sección 4.3.2 [La Instrucción Integrada Shopt], página 73, para una descripción de la opción `extdebug` para la instrucción integrada `shopt`).

`--dump-po-strings`
: Se imprime una lista de todas las cadenas entre comillas dobles en la salida estándar en el formato de archivo PO (portable object en inglés, objeto portable) de GNU `gettext`. Equivalente a `-D` excepto por el formato de salida.

`--dump-strings`
: Equivalente a `-D`.

`--help`
: Muestra un mensaje de uso en la salida estándar y finaliza exitosamente.

`--init-file` *nombre-de-archivo*
`--rcfile` *nombre-de-archivo*
: Ejecuta instrucciones de *nombre-de-archivo* (en vez de `~/.bashrc`) en un intérprete interactivo.

`--login`
: Equivalente a `-l`.

`--noediting`
: No usa la biblioteca GNU Readline (véase Capítulo 8 [Edición en Línea de Órdenes], página 123) para leer líneas de órdenes cuando el intérprete está inactivo.

`--noprofile`
: No carga el archivo de inicio de alcance global `/etc/profile` o cualquiera de los archivos de inicialización personales `~/.bash_profile`, `~/.bash_login` o `~/.profile` cuando Bash es llamado como un intérprete de acceso.

`--norc`
: No lee el archivo de inicialización `~/.bashrc` en un intérprete interactivo. Esto está activado por defecto si se llama al intérprete como `sh`.

`--posix` Cambia el comportamiento de Bash donde la operación por defecto difiera del estándar POSIX para cumplir el estándar. Esto está pensado para hacer que Bash se comporte como un componente preciso de ese estándar. Véase Sección 6.11 [Modo POSIX de Bash], página 111, para una descripción del modo POSIX de Bash.

`--restricted`
Vuelve el intérprete un intérprete restringido (véase Sección 6.10 [El Intérprete Restringido], página 110).

`--verbose`
Equivalente a `-v`. Imprime las líneas de entrada del intérprete según son leídas.

`--version`
Muestra la información de la versión para esta instancia de Bash en la salida estándar y finaliza exitosamente.

Hay varias opciones de un solo carácter que pueden ser proporcionadas durante la llamada que no están disponibles con la instrucción integrada `set`.

`-c` Lee y ejecuta instrucciones del primer argumento no opción *cadena_de_instrucción*, y finaliza. Si hay argumentos después de *cadena_de_instrucción*, se asigna el primer argumento a `$0` y se asignan los argumentos restantes a los parámetros posicionales. La asignación a `$0` establece el nombre del intérprete, que se usa en advertencias y mensajes de error.

`-i` Obliga al intérprete a ejecutarse interactivamente. Los intérpretes interactivos se describen en Sección 6.3 [Intérpretes Interactivos], página 99.

`-l` Hace que este intérprete actúe como si se hubiera llamado directamente mediante acceso. Cuando el intérprete es interactivo, esto es equivalente a iniciar un intérprete de acceso con '`exec -l bash`'. Cuando el intérprete no es interativo, serán ejecutados los archivos de inicio del intérprete de acceso. '`exec bash -l`' o '`exec bash --login`' reemplazarán el intérprete actual con un intérprete de acceso de Bash. Véase Sección 6.2 [Archivos de Inicio de Bash], página 97, para una descripción del comportamiento especial de un intérprete de acceso.

`-r` Vuelve el intérprete un intérprete restringido (véase Sección 6.10 [El Intérprete Restringido], página 110).

`-s` Si está presente esta opción, o si no quedan argumentos tras el procesado de opciones, las instrucciones son leídas de la entrada estándar. Esta opción permite que sean establecidos los parámetros posicionales al llamar a un intérprete interactivo o al leer entrada a través de una tubería.

`-D` Se imprime una lista de todas las cadenas entre comillas dobles a la entrada estándar. Estas son las cadenas que están sujetas a la traducción de idioma cuando la configuración regional actual no es `C` o `POSIX` (véase Sección 3.1.2.5 [Traducción de Localización], página 7). Esto implica la opción `-n`; no se ejecutarán instrucciones.

`[-+]O [opción_shopt]`
: *opción_shopt* es una de las opciones del intérprete aceptadas por la instrucción integrada `shopt` (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73). Si está presente *opción_shopt*, `-O` habilita el valor de esa opción; `+O` lo deshabilita. Si no se proporciona *opción_shopt*, se imprimen los nombres y valores de las opciones del intérprete aceptadas por `shopt` en la salida estándar. Si la opción de llamada es `+O`, la salida se muestra en un formato que puede ser reusado como entrada.

`--`
: Un `--` señala el fin de opciones y deshabilita el posterior procesado de opciones. Los argumentos después del `--` son tratados como nombres de archivos y argumentos.

Un intérprete de *acceso* es uno cuyo primer carácter del argumento cero es '`-`', o uno llamado con la opción `--login`.

Un intérprete *interactivo* es uno iniciado sin argumentos que no son opciones, a no ser que se especifique `-s`, sin especificar la opción `-c` y cuyas entrada y salida están ambas conectadas a terminales (como determina `isatty(3)`), o uno iniciado con la opción `-i`. Véase Sección 6.3 [Intérpretes Interactivos], página 99, para más información.

Si quedan argumentos después del procesamiento de opciones y ni la opción `-c` ni la `-s` han sido proporcionadas, se asume que el primer argumento es el nombre de un archivo que contiene instrucciones del intérprete (véase Sección 3.8 [Guiones del Intérprete], página 46). Cuando Bash es llamado de esta forma, `$0` es asignado al nombre del archivo, y los parámetros posicionales son establecidos a los argumentos restantes. Bash lee y ejecuta instrucciones de este archivo, después se cierra. El estado de salida de Bash es el estado de salida de la última instrucción ejecutada en el guion. Si no se ejecutan instrucciones, el estado de salida es 0.

6.2 Archivos de Inicio de Bash

Esta sección describe cómo ejecuta Bash sus archivos de inicio. Si cualquiera de los archivos existe pero no puede ser leído, Bash notifica un error. Las virgulillas se expanden en nombres de archivos como se describió anteriormente en Expansión de Virgulilla (véase Sección 3.5.2 [Expansión de Virgulilla], página 26).

Los intérpretes interactivos se describen en Sección 6.3 [Intérpretes Interactivos], página 99.

Llamado como un intérprete de acceso interactivo, o con `--login`

Cuando Bash es llamado como un intérprete de acceso interactivo o como un intérprete no interactivo con la opción `--login`, primero lee y ejecuta instrucciones del archivo `/etc/profile`, si ese archivo existe. Después de leer ese archivo, busca `~/.bash_profile`, `~/.bash_login` y `~/.profile`, en ese orden, y lee y ejecuta instrucciones del primero que existe y es legible. La opción `--noprofile` puede ser usada cuando se inicia el intérprete para inhibir este comportamiento.

Cuando finaliza un intérprete interactivo de acceso o un intérprete no interactivo de acceso ejecuta la instrucción integrada `exit`, Bash lee y ejecuta instrucciones del archivo `~/.bash_logout`, si existe.

Llamado como un intérprete no de acceso

Cuando se inicia un intérprete interactivo que no es un intérprete de acceso, Bash lee y ejecuta instrucciones de `~/.bashrc`, si ese archivo existe. Esto puede ser inhibido usando la opción `--norc`. La opción `--rcfile` *archivo* forzará a Bash a leer y ejecutar instrucciones de *archivo* en lugar de `~/.bashrc`.

Así, típicamente, su `~/.bash_profile` contiene la línea

```
if [ -f ~/.bashrc ]; then . ~/.bashrc; fi
```

después (o antes) de las inicializaciones específicas de acceso.

Llamado de forma no interactiva

Cuando Bash se inicia de forma no interactiva, para ejecutar un guion del intérprete, por ejemplo, busca la variable `BASH_ENV` en el entorno, expande su valor si aparece ahí y usa el valor expandido como el nombre de un archivo que leer y ejecutar. Bash se comporta como si la siguiente instrucción fuera ejecutada:

```
if [ -n "$BASH_ENV" ]; then . "$BASH_ENV"; fi
```

pero el valor de la variable `PATH` no se usa para buscar el nombre de archivo.

Como se indicó anteriormente, si se llama a un intérprete interactivo con la opción `--login`, Bash trata de leer y ejecutar instrucciones de los archivos de inicio del intérprete de acceso.

Llamado con el nombre `sh`

Si se llama a Bash con el nombre `sh`, trata de imitar el comportamiento de inicio de versiones históricas de `sh` de la manera más fiel posible, ajustándose al mismo tiempo al estándar POSIX también.

Cuando se llama como un intérprete de acceso interactivo o como un intérprete no interactivo con la opción `--login`, trata primero de leer y ejecutar instrucciones de `/etc/profile` y `~/.profile`, en ese orden. La opción `--noprofile` puede ser usada para inhibir este comportamiento. Al llamarse como un intérprete interactivo con el nombre `sh`, Bash busca la variable `ENV`, expande su valor si está definido y usa el valor expandido como el nombre de un archivo que leer y ejecutar. Puesto que un intérprete llamado con `sh` no trata de leer y ejecutar instrucciones de otros archivos de inicio, la opción `--rcfile` no tiene efecto. Un intérprete no interactivo llamado con el nombre `sh` no trata de leer otros archivos de inicio.

Al llamarse con `sh`, Bash entra al modo POSIX después de que hayan sido leídos los archivos de inicio.

Llamado en modo POSIX

Cuando se inicia Bash en modo POSIX, como con la opción de línea de órdenes `--posix`, sigue el estándar POSIX para archivos de inicio. En este modo, los intérpretes interactivos expanden la variable `ENV` y las instrucciones son leídas y ejecutadas del archivo cuyo nombre es el valor expandido. No se leen ningunos otros archivos de inicio.

Llamado por un demonio del intérprete remoto

Bash trata de determinar cuándo está siendo ejecutado con su entrada estándar conectada a una conexión de red, así como cuándo es ejecutado por el demonio remoto del intérprete,

normalmente `rshd`, o el demonio del intérprete seguro `sshd`. Si Bash determina que está siendo ejecutado de esta forma, lee y ejecuta instrucciones de `~/.bashrc`, si ese archivo existe y es legible. No hará esto al ser llamado con `sh`. La opción `--norc` puede ser usada para inhibir este comportamiento, y la opción `--rcfile` puede ser usada para forzar que sea leído otro archivo, pero ni `rshd` ni `sshd` llaman generalmente al intérprete con estas opciones o permiten que sean especificadas.

Llamado con diferentes UID/GIDs reales y efectivos

Si Bash es llamado con el id de usuario efectivo (grupo) diferente al id de usuario real (grupo) y no se proporciona la opción `-p`, no se lee ningún archivo de inicio, las funciones del intérprete no se heredan del entorno, las variables `SHELLOPTS`, `BASHOPTS`, `CDPATH` y `GLOBIGNORE`, si aparecen en el entorno, son ignoradas y el id de usuario efectivo se establece al id de usuario real. Si proporciona la opción `-p` en la invocación, el comportamiento de inicio es el mismo, pero el id de usuario efectivo no se restablece.

6.3 Intérpretes Interactivos

6.3.1 Qué es un Intérprete interactivo?

Un intérprete interactivo es uno iniciado sin argumentos de opción, a no ser que se especifique `-s`, sin especificar la opción `-c`, y cuyas entrada y salida de error están conectadas a terminales (según se determina por `isatty(3)`), o el iniciado con la opción `-i`.

Un intérprete interactivo generalmente lee de y escribe a la terminal de un usuario.

La opción de llamada `-s` puede usarse para establecer los parámetros posicionales cuando se inicia un intérprete interactivo.

6.3.2 Es este Intérprete Interactivo?

Para determinar dentro de un guion de inicio si Bash está corriendo interactivamente, compruebe el valor del parámetro especial '`-`'. Contiene `i` cuando el intérprete es interactivo. Por ejemplo:

```
case "$-" in
*i*) echo Este intérprete es interactivo ;;
*) echo Este intérprete no es interactivo ;;
esac
```

Alternativamente, los guiones de inicio pueden examinar la variable `PS1`; está sin asignar en intérpretes no intéractivos y asignada en intérpretes interactivos. Así:

```
if [ -z "$PS1" ]; then
        echo Este intérprete es no interactivo
else
        echo Este intérprete es interactivo
fi
```

6.3.3 Comportamiento del Intérprete Interactivo

Cuando el intérprete está corriendo interactivamente, cambia su comportamiento de varias formas.

1. Los archivos de inicio son leídos y ejecutados como se describe en Sección 6.2 [Archivos de Inicio de Bash], página 97.
2. El Control de Tareas (véase Capítulo 7 [Control de Tareas], página 118) está activado por defecto. Cuando el control de tareas está en efecto, Bash ignora las señales de control de tareas generadas por el teclado `SIGTTIN`, `SIGTTOU` y `SIGTSTP`.
3. Bash expande y muestra `PS1` antes de leer la primera línea de una instrucción, y expande y muestra `PS2` antes de leer la segunda y posteriores líneas de una instrucción multilínea. Bash expande y muestra `PS0` tras leer una instrucción pero antes de ejecutarla. Consulte Sección 6.9 [Controlando el Prompt], página 109, para una lista completa de secuencias de escape de cadenas de prompt.
4. Bash ejecuta los valores del conjunto de elementos del vector `PROMPT_COMMANDS` como instrucciones antes de mostrar el prompt primario, `$PS1` (véase Sección 5.2 [Variables de Bash], página 82).
5. Readline (véase Capítulo 8 [Edición en Línea de Órdenes], página 123) se usa para leer instrucciones desde la terminal del usuario.
6. Bash inspecciona el valor de la opción `ignoreeof` para `set -o` en vez de finalizar inmediatamente cuando recibe un `EOF` en su salida estándar al leer una instrucción (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68).
7. El historial de instrucciones (véase Sección 9.1 [Servicios del Historial de Bash], página 161) y la expansión de historial (véase Sección 9.3 [Interacción con el Historial], página 163) están habilitados por defecto. Bash guardará el historial de instrucciones en el archivo nombrado por `$HISTFILE` cuando finaliza un intérprete con el historial habilitado.
8. La expansión de alias (véase Sección 6.6 [Aliases], página 105) se realiza por defecto.
9. En ausencia de traps, Bash ignora `SIGTERM` (véase Sección 3.7.6 [Señales], página 45).
10. En ausencia de traps, `SIGINT` es atrapada y manejada (véase Sección 3.7.6 [Señales], página 45). `SIGINT` interrumpirá algunas instrucciones integradas del intérprete.
11. Un intérprete de acceso interactivo envía una `SIGHUP` a todas las tareas en la salida si la opción del intérprete `huponexit` ha sido habilitada (véase Sección 3.7.6 [Señales], página 45).
12. La opción de llamada `-n` es ignorada, y '`set -n`' no tiene efecto (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68).
13. Bash comprobará el correo periódicamente, dependiendo de los valores de las variables del intérprete `MAIL`, `MAILPATH` y `MAILCHECK` (véase Sección 5.2 [Variables de Bash], página 82).
14. Los errores de expansión debidos a referencias a variables sin asociar del intérprete después de que '`set -u`' haya sido habilitada no harán que el intérprete finalice (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68).
15. El intérprete no finalizará durante errores de expansión causados por *var* estando sin asignar o nula en expansiones `${var:?palabra}` (véase Sección 3.5.3 [Expansión de Parámetros del Intérprete], página 27).
16. Los errores de redirección encontrados por instrucciones integradas del intérprete no harán que el intérprete finalice.

17. Cuando se ejecuta en modo POSIX, una instrucción integrada especial que retorne un estado de error no hará que el intérprete se cierre (véase Sección 6.11 [Modo POSIX de Bash], página 111).
18. Un `exec` fallido no causará que el intérprete se cierre (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48).
19. Errores de sintaxis del analizador no harán que el intérprete finalice.
20. Está activada por defecto la corrección simple para argumentos de directorio para la instrucción integrada `cd` (consulte la descripción de la opción `cdspell` para la instrucción integrada `shopt` en Sección 4.3.2 [La Instrucción Integrada Shopt], página 73).
21. El intérprete comprobará el valor de la variable `TMOUT` y finalizará si una instrucción no es leída dentro del número de segundos especificado antes de imprimir `$PS1` (véase Sección 5.2 [Variables de Bash], página 82).

6.4 Expresiones Condicionales de Bash

Las expresiones condicionales son usadas por la instrucción compuesta `[[` y las instrucciones integradas `test` y `[`. Las instrucciones `test` y `[` determinan su comportamiento basadas en el número de argumentos; consulte las descripciones de estas instrucciones para cualquier otra acción especifica de la instrucción.

Las expresiones pueden ser unarias o binarias, y están formadas por las siguientes unarias. Las expresiones unarias se usan frecuentemente para examinar el estado de un archivo. Hay operadores de cadenas y operadores numéricos de comparación también. Bash maneja varias nombres de archivo de forma especial cuando son usados en expresiones. Si el sistema operativo en que Bash está corriendo proporciona estos archivos especiales, Bash los usará; de lo contrario, los emulará internamente con este comportamiento: si el argumento *archivo* para uno de los primarios es de la forma `/dev/fd/`*N*, entonces se comprueba el descriptor de archivo *N*. Si el argumento *archivo* para uno de los primarios es `/dev/stdin`, `/dev/stdout`, o `/dev/stderr` se comprueba el descriptor de archivo 0, 1 o 2, respectivamente.

Al usarse con `[[`, los operadores '`<`' y '`>`' ordenan lexicográficamente usando la configuración regional actual. La instrucción `test` usa la ordenación ASCII.

A no ser que se especifique lo contrario, los primarios que operan en archivo siguen enlaces simbólicos y operan en el destino del enlace, en vez de en el propio enlace.

`-a` *archivo*
: Verdadero si *archivo* existe.

`-b` *archivo*
: Verdadero si *archivo* existe y es un archivo especial de bloque.

`-c` *archivo*
: Verdadero si *archivo* existe y es un archivo especial de carácter.

`-d` *archivo*
: Verdadero si *archivo* existe y es un directorio.

`-e` *archivo*
: Verdadero si *archivo* existe.

`-f` *archivo*
: Verdadero si *archivo* existe y es un fichero normal.

`-g` *archivo*
: Verdadero si *archivo* existe y su bit set-group-id está establecido.

`-h` *archivo*
: Verdadero si *archivo* existe y es un enlace simbólico.

`-k` *archivo*
: Verdadero si *archivo* existe y su bit pegajoso está establecido.

`-p` *archivo*
: Verdadero si *archivo* existe y es una tubería con nombre (FIFO).

`-r` *archivo*
: Verdadero si *archivo* existe y es legible.

`-s` *archivo*
: Verdadero si *archivo* existe y tiene un tamaño mayor que cero.

`-t` *da* Verdadero si el descriptor de archivo *da* está abierto si se refiere a una terminal.

`-u` *archivo*
: Verdadero si *archivo* existe y su bit set-user-id está establecido.

`-w` *archivo*
: Verdadero si *archivo* existe y es escribible.

`-x` *archivo*
: Verdadero si *archivo* existe y es ejecutable.

`-G` *archivo*
: Verdadero si *archivo* existe y está poseído por el id de grupo efectivo.

`-L` *archivo*
: Verdadero si *archivo* existe y es un enlace simbólico.

`-N` *archivo*
: Verdadero si *archivo* existe y ha sido modificado desde que fue leído por última vez.

`-O` *archivo*
: Verdadero si *archivo* existe y está poseído por el id de usuario efectivo.

`-S` *archivo*
: Verdadero si *archivo* existe y es un socket.

archivo1 `-ef` *archivo2*
: Verdadero si *archivo1* y *archivo2* se refieren a los mismos números de dispositivo y de inodo.

archivo1 `-nt` *archivo2*
: Verdadero si *archivo1* es más nuevo (según la fecha de modificación) que *archivo2*, o si *archivo1* existe y *archivo2* no.

archivo1 `-ot` *archivo2*
: Verdadero si *archivo1* es más viejo que *archivo2*, o si *archivo2* existe y *archivo1* no.

`-o` *nombre-de-opc*
: Verdadero si la opción del intérprete *nombre-de-opc* está habilitada. La lista de opciones aparece en la descripción de la opción `-o` de la instrucción integrada `set` (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68).

`-v` *nombre-de-var*
: Verdadero si la variable del intérprete *nombre-de-var* está establecida (se le ha asignado un valor).

`-R` *nombre-de-var*
: Verdadero si la variable del intérprete *nombre-de-var* está establecida y es una referencia de nombre.

`-z` *cadena* Verdadero si la longitud de *cadena* es cero.

`-n` *cadena*
cadena Verdadero si la longitud de *cadena* es distinta de cero.

cadena1 `==` *cadena2*
cadena1 `=` *cadena2*
: Verdadero si las cadenas son iguales. Cuando se usa con la instrucción `[[`, esto realiza la coincidencia de patrones como se describió anteriormente (véase Sección 3.2.5.2 [Construcciones Condicionales], página 12).

  '`=`' debería usarse con la instrucción `test` para la conformidad con POSIX.

cadena1 `!=` *cadena2*
: Verdadero si las cadenas no son iguales.

cadena1 `<` *cadena2*
: Verdadero si *cadena1* se ordena lexicográficamente antes que *cadena2*.

cadena1 `>` *cadena2*
: Verdadero si *cadena1* se ordena lexicográficamente después que *cadena2*.

arg1 `OP` *arg2*
: `OP` es uno de '`-eq`', '`-ne`', '`-lt`', '`-le`', '`-gt`' o '`-ge`'. Estos operadores aritméticos binarios devuelven verdadero si *arg1* es igual a, no igual a, menor que, menor que o igual a, mayor que, mayor que o igual a *arg2*, respectivamente. *Arg1* y *arg2* pueden ser enteros positivos o negativos. Cuando se usa con la instrucción `[[`, *Arg1* y *Arg2* se evalúan como expresiones aritméticas (véase Sección 6.5 [Aritmética del Intérprete], página 103).

6.5 Aritmética del Intérprete

El intérprete permite evaluar expresiones aritméticas, como una de las expansiones del intérprete o usando la instrucción compuesta `((`, la instrucción integrada `let` o la opción `-i` de la instrucción integrada `declare`.

La evaluación se hace en enteros de anchura fija sin comprobar por desbordamiento, aunque se atrapa la división entre 0 y se marca como error. Los operadores y su precedencia, asociatividad y valores son los mismos que en lenguaje C. La siguiente lista de operadores está agrupada en niveles de operadores de igual precedencia. Los niveles son listados en orden de precedencia decreciente.

id`++` *id*`--` postincremento y postdecremento de variable

`++`*id* `--`*id* preincremento y predecremento de variable

`- +` menos y más unarios

`! ~` negación lógica y a nivel de bit

`**` exponenciación

`* / %` multiplicación, división, resto

`+ -` adición, sustracción

`<< >>` desplazamientos de bit a izquierda y derecha

`<= >= < >` comparación

`== !=` igualdad y desigualdad

`&` AND a nivel de bit

`^` OR exclusivo a nivel de bit

`|` OR a nivel de bit

`&&` AND lógico

`||` OR lógico

`expr ? expr : expr`
operador condicional

`= *= /= %= += -= <<= >>= &= ^= |=`
asignación

`expr1 , expr2`
coma

Las variables del intérprete se permiten como operandos; la expansión de parámetro se realiza antes de que se evalúe la expresión. Dentro de una expresión, las variables del intérprete también pueden ser referenciadas por nombre sin usar la sintaxis de expansión de parámetros. Una variable del intérprete que es nula o no asignada evalúa a 0 cuando se referencia por nombre sin usar la sintaxis de expansión de parámetro. El valor de una variable se evalúa como una expresión aritmética cuando es referenciado o cuando a una variable a la que se le ha dado el atributo *integer* usando '`declare -i`' se le asigna un valor. Un valor nulo evalúa a 0. Una variable del intérprete no tiene que tener su atributo *integer* activado para usarse en una expresión.

Las constantes enteras siguen la definición del lenguaje C, sin sufijos o caracteres constantes. Las constantes con un 0 inicial se interpretan como números octales. Un '`0x`' o '`0X`' denota hexadecimal. De los contrario, los números toman la forma [*base*`#`]*n*, donde la *base* opcional es un número hexadecimal entre 2 y 64 que representa la base aritmética, y *n* es un número en esa base. Si se omite *base*`#`, entonces se usa la base 10. Al especificar *n*, si se requieren no-dígitos, los dígitos mayores que nuevo se representan por las letras minúsculas, las letras mayúsculas, '`@`' y '`_`', en ese orden. Si *base* es menor o igual a 36, las letras minúsculas y mayúsculas pueden ser usadas indistintamente para representar números entre el 10 y el 35.

Los operadores se evalúan en orden de precedencia. Las subexpresiones en paréntesis se evalúan primero y pueden sobrescribir las reglas de precedencia anteriores.

6.6 Aliases

Los *aliases* permiten que una cadena sea sustituida por una palabra cuando se usa como la primera palabra de una instrucción simple. El intérprete mantiene una lista de aliases que puede ser establecida y eliminada con las instrucciones integradas `alias` y `unalias`.

La primera palabra de cada instrucción simple, si no está entrecomillada, se comprueba para ver si tiene un alias. En ese caso, ese palabra es reemplazada por el texto del alias. Los caracteres '`/`', '`$`', '`‘`', '`=`' y cualquiera de los metacaracteres o caracteres de entrecomillado del intérprete listados anteriormente no pueden aparecer en un nombre de alias. El texto de reemplazo puede contener cualquier entrada de intérprete válida, incluyendo metacaracteres del intérprete. La primera palabra de texto de reemplazo se comprueba por aliases, pero una palabra que es idéntica a un alias que está siendo expandido no se expande una segunda vez. Esto significa que uno puede crear el alias `ls` para `"ls -F"`, por ejemplo, y Bash no trata de expandir recursivamente el texto de reemplazo. Si el último carácter del valor del alias es un *blanco*, entonces la siguiente palabra de instrucción que sigue al alias también se comprueba para la expansión de alias.

Los alias son creados y listados con la instrucción `alias`, y eliminados con la instrucción `unalias`.

No hay mecanismo para usar argumentos en el texto de reemplazo, como en `csh`. Si se necesitan argumentos, se debería usar una función del intérprete (véase Sección 3.3 [Funciones del Intérprete], página 19).

Los aliases no se expanden cuando el intérprete no es interactivo, a no ser que esté habilitada la opción del intérprete `expand_aliases` usando `shopt` (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73).

Las reglas concernientes a la definición y uso de aliases son en cierto modo confusas. Bash siempre lee al menos una línea completa de entrada, y todas las líneas que forman una instrucción compuesta, antes de ejecutar cualquiera de las instrucciones en esa línea o la instrucción compuesta. Los aliases se expanden cuando se lee una instrucción, no cuando se ejecuta. Por lo tanto, una definición de alias que aparece en la misma línea que otra instrucción no tiene efecto hasta que la siguiente línea de entrada es leída. Las instrucciones que siguen la definición del alias en esa línea no se ven afectadas por el nuevo alias. Este comportamiento es también un problema cuando se ejecutan funciones. Los aliases se expanden cuando se lee una definición de función, no cuando se ejecuta la función, porque una definición de función es en sí una instrucción. Como consecuencia, los aliases definidos en una función no están disponibles hasta después de que la función sea ejecutada. Para estar seguro, ponga siempre las definiciones de alias en una línea separada, y no use `alias` en instrucciones compuestas.

Para casi cualquier propósito, se prefieren las funciones del intérprete a los aliases.

6.7 Vectores

Bash proporciona variables de vectores indexadas y asociadas de una dimensión. Cualquier variable puede ser usada como un vector asociativo; la instrucción integrada `declare` declarará explícitamente un vector. No hay límite máximo sobre el tamaño de un vector, ni ningún requerimiento de que los miembros estén indexados o asignados contiguamente. Los vectores indexados son referenciados usando enteros (incluyendo expresiones aritméticas

[véase Sección 6.5 [Aritmética del Intérprete], página 103]) y están basados en cero; los vectores asociativos usan cadenas arbitrarias. A no ser que se indique lo contrario, los indices de vectores indexados deben ser enteros no negativos.

Un vector indexado se crea automáticamente si se asigna alguna variable usando la sintaxis

```
nombre[subíndice]=valor
```

El *subíndice* se trata como una expresión aritmética que debe evaluar a un número. Para declarar un vector explícitamente, use

```
declare -a nombre
```

La sintaxis

```
declare -a nombre[subíndice]
```

también se acepta; el *subíndice* es ignorado.

Los vectores asociativos son creados usando

```
declare -A nombre
```

Los atributos pueden ser especificados para una variable de vector usando las instrucciones integradas `declare` y `readonly`. Cada atributo se aplica a todos los miembros del vector.

Los vectores son asignados usando asignaciones compuestas de la forma

```
nombre=(value1 value2 ... )
```

donde cada *valor* puede ser de la forma `[subíndice]=`*cadena*. Las asignaciones de vectores indexados no requieren nada más que una *cadena*. Al asignar a vectores indexados, si se proporciona el subíndice opcional, ese índice se asigna; de lo contrario, el índice del elemento asignado es el último índice asignado por la cláusula más uno. El indexado comienza por cero.

Cada *valor* en la lista experimenta todas las expansiones del intérprete descritas anteriormente (véase Sección 3.5 [Expansiones del Intérprete], página 24).

Al asignar a un vector asociativo, las palabras en una asignación compuesta pueden ser o bien cláusulas de asignación, para las cuales es necesario el subíndice, o bien una lista de palabras que es interpretada como una secuencia de claves y valores que se alternan: *nombre*=(*clave1 valor1 clave2 valor2* ...). Estos se tratan de forma idéntica a *nombre*=([*clave1*]=*valor1* [*clave2*]=*valor2* ...). La primera palabra en la lista determina cómo se interpretan las palabras restantes; todas las asignaciones en una lista deben ser del mismo tipo. Al usar pares de clave-valor, las claves no pueden faltar o estar vacías; un último valor final es tratado como la cadena vacía.

Esta sintaxis también es aceptada por la instrucción integrada `declare`. A los elementos individuales de vector también se puede asignar usando la sintaxis `nombre[subíndice]=valor` introducida anteriormente.

Al asignar a un vector indexado, si *nombre* está subindexado por un número negativo, ese número se interpreta como relativo a uno mayor que el índice máximo de *nombre*, de forma que los índices negativos cuentan hacia tras desde el final del vector, y un índice de -1 hace referencia al último elemento.

Se puede referenciar cualquier elemento de un vector usando `${nombre[subíndice]}`. Las llaves se requieren para evitar conflictos con los operadores de expansión de nombre de

archivo del intérprete. Si *subíndice* es '`@`' o '`*`', la palabra se expande a todos los miembros del vector *nombre*. Estos subíndices difieren solo cuando la palabra aparece dentro de comillas dobles. Si la palabra está entre comillas dobles, `${nombre[*]}` se expande a una única palabra con el valor de cada miembro del vector separado por el primer carácter de la variable `IFS`, y `${nombre[@]}` expande cada elemento de *nombre* a una palabra separada. Cuando no hay miembros del vector, `${nombre[@]}` se expande a nada. Si la expansión entre comillas dobles ocurre dentro de una palabra, la expansión del primer parámetro se une con la parte inicial de la palabra original, y la expansión del último parámetro se une con la última parte de la palabra original. Esto es análogo a la expansión de los parámetros especiales '`@`' y '`*`'. `${#nombre[subíndice]}` se expande a la longitud de `${nombre[subíndice]}`. Si *subíndice* es '`@`' o '`*`', la expansión es el número de elementos del vector. Si el *subíndice* usado para referenciar un elemento de un vector indexado evalúa a un número menor que cero, se interpreta como relativo a uno mayor que el índice máximo del vector, de forma que los índices negativos cuentan hacia tras desde el final del vector y un índice de -1 hace referencia al último elemento.

Referenciar una variable de vector sin un subíndice es equivalente a referenciar con un subíndice de 0. Cualquier referencia a una variable usando un subíndice válido es legal, y `bash` creará un vector si es necesario.

Una variable de vector se considera establecida si se ha asignado un valor a un subíndice. La cadena nula es un valor válido.

Es posible obtener las claves (índices) de un vector así como los valores. ${!*nombre*[@]} y ${!*nombre*[*]} se expanden a los índices asignados en una variable de vector *nombre*. El tratamiento en comillas dobles es similar a la expansión de los parámetros especiales '`@`' y '`*`' dentro de comillas dobles.

La instrucción integrada `unset` se usa para destruir vectores. `unset nombre[subíndice]` destruye el elemento de vector en el índice *subíndice*. Los subíndices negativos para vectores indexados se interpretan como se describe arriba. Eliminar el último elemento de una variable de vecor no elimina la variable. `unset nombre`, donde *nombre* es un vector, elimina el vector completo. Un subíndice de '`*`' o '`@`' también elimina el vector completo.

Al usar un nombre de variable con un subíndice como argumento para una instrucción, como con `unset`, sin usar la sintaxis de expansión de palabra descrita anteriormente, el argumento está sujeto a la expansión de nombre de archivo del intérprete. Si no se desea la expansión de nombre de archivo, el argumento debería ser entrecomillado.

Cada una de las instrucciones integradas `declare`, `local` y `readonly` aceptan una opción `-a` para especificar un vector indexado y una opción `-A` para especificar un vector asociativo. Si se proporcionan ambas opciones, `-A` toma precedencia. La instrucción integrada `read` acepta una opción `-a` para asignar una lista de palabras leídas de la entrada estándar a un vector, y puede leer valores de la entrada estándar en elementos de vector individuales. Las instrucciones integradas `set` y `declare` muestran valores de vector de una forma que permiten que sean reutilizados como entrada.

6.8 La Pila de Directorios

La pila de directorios es una lista de directorios visitados recientemente. La instrucción integrada `pushd` añade directorios a la pila mientras cambia el directorio actual, y la instrucción integrada `popd` elimina los directorios especificados de la pila y cambia el directorio actual

al directorio eliminado. La instrucción integrada `dirs` muestra los contenidos de la pila de directorios. El directorio actual es siempre el superior de la pila de directorios.

Los contenidos de la pila de directorios también son visibles como el valor de la variable del intérprete `DIRSTACK`.

6.8.1 Instrucciones Integradas de la Pila de Directorios

`dirs`

```
dirs [-clpv] [+N | -N]
```

Muestra la lista de los directorios actualmente recordados. Los directorios son añadidos a la lista con la instrucción `pushd`; la instrucción `popd` elimina directorios de la lista. El directorio actual es siempre el primer directorio en la pila.

`-c` Limpia la pila de directorios eliminando todos los elementos.

`-l` Produce un listado usando nombres de ruta completos; el formato de listado predeterminado usa una virgulilla para denotar el directorio principal.

`-p` Hace que `dirs` imprima la pila de directorios con una entrada por línea.

`-v` Hace que `dirs` imprima la pila de directorios con una entrada por línea, prefijando cada entrada con su índice en la pila.

`+`*N* Muestra el directorio número *N* (contando desde la izquierda de la lista imprimida por `dirs` cuando es llamado sin opciones), empezando por cero.

`-`*N* Muestra el directorio número *N* (contando desde la derecha de la lista imprimida por `dirs` cuando es llamado sin opciones), empezando por cero.

`popd`

```
popd [-n] [+N | -N]
```

Cuando no se pasan argumentos, `popd` elimina el directorio superior de la pila y realiza un `cd` al nuevo directorio superior. Los elementos están numerados desde 0 empezando por el primer directorio listado con `dirs`; esto es, `popd` es equivalente a `popd +0`.

`-n` Suprime el cambio de directorio normal al eliminar directorios de la pila, de forma que solo se manipula la pila.

`+`*N* Elimina el directorio número *N* (contando desde la izquierda de la lista imprimida por `dirs`), empezando por cero.

`-`*N* Elimina el directorio número *N* (contando desde la derecha de la lista imprimida por `dirs`), empezando por cero.

`pushd`

```
pushd [-n] [+N | -N | dir]
```

Guarda el directorio actual en la parte superior de la pila de directorios y después hace un `cd` a *dir*. Sin argumentos, `pushd` intercambia los dos directorios superiores y convierte el nuevo superior en el directorio actual.

`-n` Suprime el cambio normal de directorio al rotar o añadir directorios a la pila, de forma que solo se manipula la pila.

`+`*N* Trae el directorio número *N* (contando desde la izquierda de la lista imprimida por `dirs`, empezando por cero) a la parte superior de la lista rotando la pila.

`-`*N* Trae el directorio número *N* (contando desde la derecha de la lista imprimida por `dirs`, empezando por cero) a la parte superior de la lista rotando la pila.

dir Hace que *dir* sea el superior de la pila, volviéndolo el nuevo directorio actual como si hubiera sido proporcionado como un argumento para la instrucción integrada `cd`.

6.9 Controlando el Prompt

Bash examina el valor de la variable de vector `PROMPT_COMMANDS` justo antes de mostrar cada prompt primario. Si hay algunos elementos asignados y no nulos en `PROMPT_COMMANDS`, Bash ejecuta cada valor en orden número, exactamente como si hubiera sido escrito en la.

Además, la siguiente tabla describe los caracteres especiales que pueden aparecer en las variables de prompt `PS0`, `PS1`, `PS2` y `PS4`:

`\a` Un carácter de timbre.

`\d` La fecha, en formato "DíaSemana Mes Día" (p. ej. "Mar May 26").

`\D{`*formato*`}`
El *formato* es pasado a `strftime`(3) y el resultado es insertado en la cadena del prompt; un *formato* vacío resulta en una representación del tiempo específica de región. Se requieren las llaves.

`\e` Un carácter de escape.

`\h` El nombre del anfitrión, hasta el primer '.'.

`\H` El nombre del anfitrión.

`\j` El número de tareas actualmente gestionadas por el intérprete.

`\l` El nombre base del nombre de dispositivo de la terminal del intérprete.

`\n` Una nueva línea.

`\r` Un retorno de carro.

`\s` El nombre del intérprete, el nombre base de `$0` (la porción que sigue a la última barra).

`\t` La hora, en formato HH:MM:SS de 24 horas.

`\T` La hora, en formato HH:MM:SS de 12 horas.

`\@` La hora, en formato am/pm de 12 horas.

`\A` La hora, en formato HH:MM de 24 horas.

`\u` El nombre de usuario del usuario actual.

`\v` La versión de Bash (p. ej., 2.00)

`\V` La publicación de Bash, versión + nivel de parche (p. ej., 2.00.0)

`\w` El directorio de trabajo actual, con `$HOME` abreviado con una virgulilla (usa la variable `$PROMPT_DIRTRIM`).

`\W` El nombre base de `$PWD`, con `$HOME` abreviado con una virgulilla.

`\!` El número de historial de esta instrucción.

`\#` El número de instrucción de esta instrucción.

`\$` `#` si el uid efectivo es 0, sino `$`.

`\`*nnn* El carácter cuyo código ASCII es el valor octal *nnn*.

`\\` Una barra invertida.

`\[` Empieza una secuencia de caracteres no imprimibles. Esto podría usarse para incrustar una secuencia de control de terminal en el prompt.

`\]` Termina una secuencia de caracteres no imprimibles.

El número de instrucción y el número de historial son normalmente diferentes: el número de historial de una instrucción es su posición en la lista del historial, que puede incluir instrucciones recuperadas del archivo del historial (véase Sección 9.1 [Servicios del Historial de Bash], página 161), mientras que el número de instrucción es la posición en la secuencia de instrucciones ejecutada durante la sesión actual del intérprete.

Después de que la cadena sea decodificada, se expande mediante la expansión de parámetro, la sustitución de instrucción, la expansión aritmética y eliminación de comillas, sujeta al valor de la opción del intérprete `promptvars` (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73). Esto puede tener efectos secundarios inesperados si las porciones de la cadena aparecen dentro de sustituciones de instrucciones o contienen caracteres especiales para la expansión de palabras.

6.10 El Intérprete Restringido

Si Bash se inicia con el nombre `rbash`, o la opción `--restricted` o `-r` se proporciona en la llamada, el intérprete se vuelve restringido. Un intérprete restringido se usa para establecer un entorno más controlado que el intérprete estándar. Un intérprete restringido se comporta idénticamente a `bash` con la excepción de que lo siguiente es rechazado o no es realizado:

- Cambiar directorios con la instrucción integrada `cd`.
- Establecer o eliminar los valores de las variables `SHELL`, `PATH`, `HISTFILE`, `ENV` o `BASH_ENV`.
- Especificar nombres de instrucciones que contienen barras.
- Especificar un nombre de archivo que contiene una barra como un argumento para la instrucción integrada `..`

- Especificar un nombre de archivo que contiene una barra como argumento para la instrucción integrada `history`.
- Especificar un nombre de archivo que contiene una barra como un argumento para la opción `-p` de la opción integrada `hash`.
- Importar definiciones de función del entorno del intérprete durante el inicio.
- Analizar el valor de `SHELLOPTS` desde el entorno del intérprete al inicio.
- Redirigir salida usando los operadores de redirección '`>`', '`>|`', '`<>`', '`>&`', '`&>`' y '`>>`'.
- Usar la instrucción integrada `exec` para reemplazar el intérprete con otra instrucción.
- Añadir o eliminar instrucciones integradas con las opciones `-f` y `-d` de la instrucción integrada `enable`.
- Usar la instrucción integrada `enable` para habilitar instrucciones integradas deshabilitadas.
- Especificar la opción `-p` para la instrucción integrada `command`.
- Desactivar el modo restringido con '`set +r`' o '`set +o restricted`'.

Estas restricciones son impuestas tras leerse los archivos de inicio.

Cuando se ejecuta una instrucción que resulta ser un guion del intérprete (véase Sección 3.8 [Guiones del Intérprete], página 46), `rbash` desactiva cualquier restricción en el intérprete creado para ejecutar el guion.

El modo restringido del intérprete es solo un componente de un entorno restringido útil. Debería acompañarse de asignar `PATH` a un valor que permita la ejecución de solo unas pocas instrucciones verificadas (instrucciones que permiten escapes del intérprete son particularmente vulnerables), dejando al usuario en un directorio sin permiso de escritura más que su directorio actual después del inicio de sesión, no permitiendo al intérprete restringido ejecutar guiones del intérprete y limpiando el entorno de variables que hacen que las instrucciones modifiquen su comportamiento (p. ej., `VISUAL` o `PAGER`).

Los sistemas modernos proporcionan más formas seguras de implementar un entorno restringido, como `jaulas`, `zonas`, o `contenedores`.

6.11 Modo POSIX de Bash

Empezar Bash con la opción de línea de órdenes `--posix` o ejecutar '`set -o posix`' mientras Bash está corriendo hará que Bash se adhiera más fielmente al estándar POSIX cambiando el comportamiento para cumplir el especificado por POSIX en áreas donde lo predeterminado de Bash difiera.

Cuando se llama como `sh`, Bash entra al modo POSIX después de leer los archivos de inicio.

La siguiente lista es lo que se cambia cuando el 'modo POSIX' tiene efecto:

1. Bash se asegura de que la variable `POSIXLY_CORRECT` esta habilitada.
2. Cuando una instrucción en la tabla hash ya no existe, Bash volverá a inspeccionar `$PATH` para encontrar la nueva ubicación. Esto también está disponible con '`shopt -s checkhash`'.
3. Bash no insertará una instrucción sin el bit de ejecución asignado en la tabla hash, incluso si lo devuelve como un resultado (desesperado) de una búsqueda en `$PATH`.

4. El mensaje imprimido por el código de control de tareas y las instrucciones integradas cuando una tarea finaliza con un estado distinto a cero es 'Hecho(estado)'.
5. El mensaje imprimido por el código de control de tareas y las instrucciones integradas cuando una tarea es detenida es 'Detenido(*nombre-señal*)', donde *nombre-señal* es, por ejemplo, `SIGTSTP`.
6. La expansión de alias está siempre activada, incluso en intérpretes no interactivos.
7. Las palabras reservadas que aparecen en un contexto donde se reconocen las palabras reservadas no experimentan la expansión de alias.
8. Las expansiones POSIX `PS1` and `PS2` de '`!`' para el número de historial y '`!!`' a '`!`' son habilitadas, y la expansión de parámetro se realiza en los valores de `PS1` y `PS2` sin importar la configuración de la opción `promptvars`.
9. Los archivo de inicio POSIX son ejecutados (`$ENV`) en lugar de los archivos normales de Bash.
10. La expansión de virgulilla solo se realiza en asignaciones que preceden un nombre de instrucción, en vez de en todas las sentencias de asignación en la línea.
11. El archivo predeterminado del historial es `~/.sh_history` (este es el valor predeterminado de `$HISTFILE`).
12. Los operadores de redirección no realizan la expansión de nombre de archivo en la palabra en la redirección a no ser que el intérprete sea interactivo.
13. Los operadores de redirección no realizan división de palabras en la palabra en la redirección.
14. Los nombres de funciones deben ser `nombre`s válidos del intérprete. Es decir, no pueden contener caracteres distintos de letras, dígitos, barras bajas y no pueden empezar con un dígito. Declarar una función con un nombre inválido causa un error de sintaxis fatal en intérpretes no interactivos.
15. Los nombres de funciones no pueden ser los mismos que una de las instrucciones integradas especiales POSIX.
16. Las instrucciones integradas especiales POSIX son encontradas antes que las funciones del intérprete durante la búsqueda de instrucción.
17. Al imprimir definiciones de función del intérprete (p. ej., por `type`), Bash no imprime la palabra clave `función`.
18. Las virgulillas literales que aparecen como el primer carácter en elementos de la variable `PATH` no son expandidas como se describe anteriormente bajo Sección 3.5.2 [Expansión de Virgulilla], página 26.
19. La palabra reservada `time` puede ser usada por sí misma como una instrucción. Cuando se usa de esta forma, muestra las estadísticas de tiempo para el intérprete y sus hijos completados. La variable `TIMEFORMAT` controla el formato de la información de tiempo.
20. Al analizar y expandir una expansión ${...} que aparece dentro de comillas dobles, las comillas simples ya no son especiales y no pueden usarse para entrecomillar una llave de cierre u otro carácter especial, a no ser que el operador sea uno de aquellos definidos para realizar la eliminación de patrón. En este caso, no tienen que aparecer como parejas emparejadas.
21. El analizador no reconoce `time` como una palabra reservada si el siguiente símbolo empieza por un '`-`'.

22. El carácter '`!`' no introduce la expansión del historial dentro de una cadena de comillas dobles, incluso si la opción `histexpand` está habilitada.
23. Si una instrucción integrada especial POSIX devuelve un estado de error, finaliza un intérprete no interactivo. Los errores fatales son aquellos listados en el estándar POSIX, e incluyen cosas como pasar opciones incorrectas, errores de redirección, errores de asignación de variables para asignaciones que preceden al nombre de la instrucción y demás.
24. Un intérprete no interactivo finaliza con un estado de error si un error de asignación de variable ocurre cuando ningún nombre de instrucción sigue a las sentencias de asignación. Un error de asignación de variable ocurre, por ejemplo, al intentar asignar un valor a una variable de solo lectura.
25. Un intérprete no interactivo finaliza con un estado de error si un error de asignación de variable ocurre en una sentencia de asignación que precede a una instrucción integrada especial, pero no con cualquier otra instrucción simple.
26. Un intérprete no interactivo finaliza con un estado de error si la variable de iteración en una sentencia `for` o la variable de selección en una sentencia `select` es una variable de solo lectura.
27. Los intérpretes no interactivos finalizan si *nombre-de-archivo* en `.` *nombre-de-archivo* no se encuentra.
28. Los intérpretes no interactivos finalizan si un error de sintaxis en una expansión aritmética resulta en una expresión inválida.
29. Los intérpretes no interactivos finalizan si ocurre un error de expansión de parámetro.
30. Los intérpretes no interactivos finalizan si hay un error de sintaxis en un guion leído con las instrucciones integradas `.` o `source`, o en una cadena procesada por la instrucción integrada `eval`.
31. Aunque la indirección de variable está disponible, no puede ser aplicada a los parámetros especiales '`#`' y '`?`'.
32. Al expandir el parámetro especial '`*`' en un contexto de patrón donde la expansión está entre comillas dobles no trata el `$*` como si estuviera entre comillas dobles.
33. Las sentencias de asignación que preceden a las instrucciones integradas especiales POSIX persisten en el entorno del intérprete después de que la instrucción integrada se completa.
34. La instrucción integrada especial `command` no evita que las instrucciones integradas tomen sentencias de asignación como argumentos de expandirlos como sentencias de asignación; cuando no está en modo POSIX, las instrucciones integradas de asignación pierden sus propiedades de expansión de sentencias de asignación al ser precedidas por `command`.
35. La instrucción integrada `bg` usa el formato requerido para describir cada tarea ubicada en segundo plano, que no incluye una indicación de si la tarea es la tarea actual o previa.
36. La salida de '`kill -l`' imprime todos los nombres de señal en una sola línea, separados por espacios, sin el prefijo '`SIG`'.
37. La instrucción integrada `kill` no acepta nombres de señal con un prefijo '`SIG`'.

38. Las instrucciones integradas `export` y `readonly` muestran su salida en el formato requerido por POSIX.
39. La instrucción integrada `trap` muestra nombres de señales sin el `SIG` inicial.
40. La instrucción integrada `trap` no comprueba el primer argumento en busca de una posible especificación de señal y revierte el manejo de tareas a la disposición original si está, a no ser que el argumento consista exclusivamente de dígitos y sea un número de señal válido. Si los usuarios quieren restablecer el manejador para una señal dada a la disposición original, deben usar '`-`' como el primer argumento.
41. `trap -p` muestra señales cuyas disposiciones están asignadas a SIG_DFL y aquellas que fueron ignoradas cuando se inició el intérprete.
42. Las instrucciones integradas `.` y `source` no buscan el directorio actual en busca del argumento de nombre de archivo si no es encontrado explorando `PATH`.
43. Activar el modo POSIX tiene el efecto de ajustar la opción `inherit_errexit`, de forma que los subintérpretes creados para ejecutar sustituciones de instrucciones heredan el valor de la opción `-e` del intérprete padre. Cuando no está activada la opción `inherit_errexit`, Bash limpia la opción `-e` en tales subintérpretes.
44. Habilitar el modo POSIX tiene el efecto de activar la opción `shift_verbose`, para que los argumentos numéricos de `shift` que excedan el número de parámetros posicionales produzca un mensaje de error.
45. Cuando la instrucción integrada `alias` muestra definiciones de aliases, no las muestra con un '`alias` ' inicial a no ser que se proporcione la opción `-p`.
46. Cuando instrucción integrada `set` es llamada sin opciones, no muestra los nombres y las definiciones de las funciones del intérprete.
47. Cuando la instrucción integrada `set` es llamada sin opciones, muestra valores de variables sin comillas, a no ser que contengan metacaracteres del intérprete, incluso si el resultado contiene caracteres no imprimibles.
48. Cuando la instrucción integrada `cd` es llamada en modo *logical*, y el nombre de ruta construido de `$PWD` y el nombre de directorio proporcionado como un argumento no se refiere a un directorio existente, `cd` fallará en vez de recurrir al modo *physical*.
49. Cuando la instrucción integrada builtin no puede cambiar de directorio porque la longitud del nombre de ruta construido a partir de `$PWD` y el nombre de directorio proporcionado como argumento excede *PATH_MAX* cuando se expanden todos los enlaces simbólicos, `cd` fallará en vez de intentar usar solo el nombre de directorio proporcionado.
50. La instrucción integrada `pwd` verifica que el valor que imprime es el mismo que el directorio actual, incluso si no se le pide que compruebe el sistema de ficheros con la opción `-P`.
51. Al listar el historial, la instrucción integrada `fc` no incluye una indicación de si ha sido modificada o no una entrada del historial.
52. El editor predeterminado usado por `fc` es `ed`.
53. Las instrucciones integradas `type` y `command` no presentan un archivo no ejecutable como encontrado, aunque el intérprete tratará de ejecutar tal archivo si es el único archivo así llamado encontrado en `$PATH`.
54. El modo de edición `vi` llamará al editor `vi` directamente cuando se ejecute la instrucción '`v`', en vez de comprobar `$VISUAL` y `$EDITOR`.

55. Cuando está habilitada la opción `xpg_echo`, Bash no trata de interpretar los argumentos de `echo` como opciones. Cada argumento se muestra, después de que los caracteres de escape son convertidos.
56. La instrucción integrada `ulimit` usa un tamaño de bloque de 512 bytes para las opciones `-c` y `-f`.
57. La llegada de `SIGCHLD` cuando una trap está establecida en `SIGCHLD` no interrumpe la instrucción integrada `wait` y hace que retorne inmediatamente. La instrucción trap es ejecutada una vez por cada hijo que finalice.
58. La instrucción integrada `read` puede ser interrumpida por una señal para la cual ha sido establecida un trap. Si Bash recibe una señal atrapada mientra ejecuta `read`, el manejador de trap se ejecuta y `read` devuelve un estado de error mayor que 128.
59. Bash elimina el estado de un proceso en segundo plano finalizado de la lista de tales estados después de que la instrucción integrada `wait` sea usada para obtenerlo.

Hay otro comportamiento POSIX que Bash no implementa por defecto incluso en modo POSIX. Específicamente:

1. La instrucción integrada `fc` comprueba `$EDITOR` como un programa para editar entrada de historial si `FCEDIT` está sin asignar, en vez de recurrir directamente a `ed`. `fc` usa `ed` si `EDITOR` está sin asignar.
2. Como se indica anteriormente, Bash requiere que la opción `xpg_echo` esté activada para la instrucción integrada `echo` para ser completamente conforme.

Bash puede configurarse para que sea conforme con POSIX por defecto, especificando la `--enable-strict-posix-default` para `configure` al construir (véase Sección 10.8 [Funcionalidades Opcionales], página 170).

6.12 Modo de Compatibilidad del Intérprete

Bash-4.0 introdujo el concepto de un 'nivel de compatibilidad del intérprete', especificado como una serie de opciones de la instrucción integrada shopt (`compat31`), `compat32`, `compat40`, `compat41` y demás). Solo hay un nivel de compatibilidad actual cada opción es mutuamente excluyente. El modo de compatibilidad está pensado para permitir a los usuarios seleccionar un comportamiento de versiones anteriores que es incompatible con nuevas versiones mientras que migran guiones para usar funcionalidades y comportamientos actuales.

Esta sección no menciona un comportamiento que es estándar para una versión particular (p. ej., asignar `compat32` significa que entrecomillar la parte derecha del operador de coincidencia de expresiones regulares entrecomilla caracteres especiales de expresiones regulares en la palabra, lo que es el comportamiento predeterminado en bash-3.2 y superiores).

Si un usuario habilita, digamos, `compat32`, podría afectar al comportamiento de otros niveles de compatibilidad hasta e incluyendo el nivel de compatibilidad actual. La idea es que cada nivel de compatibilidad controle un comportamiento que cambió en esa versión de Bash, pero ese comportamiento puede que haya estado presente en versiones anteriores. Por ejemplo, el cambio para usar comparaciones basadas en la configuración regional con la instrucción `[[` apareció en bash-4.1, y las versiones anteriores usaban comparaciones basadas en ASCII, por lo que habilitar `compat32` también habilitará comparaciones basadas en ASCII. Esa granularidad puede no ser suficiente para todos los usos, y como resultado los

usuarios deberían emplear los niveles de compatibilidad cuidadosamente. Lea la documentación sobre una funcionalidad particular para descubrir el comportamiento actual.

Bash-4.3 introdujo una nueva variable del intérprete: `BASH_COMPAT`. El valor asignado a esta variable (una número de versión decimal como 4.2 o un entero correspondiente a la opción `compat`*NN* como 42) determina el nivel de compatibilidad.

Desde bash-4.4, Bash a empezado a dejar de mantener niveles de compatibilidad más viejos. Finalmente, las opciones serán eliminadas en favor de `BASH_COMPAT`.

Bash-5.0 es la última versión para la que habrá una opción individual shopt para la anterior versión. Los usuarios deberían usar `BASH_COMPAT` en bash-5.0 y versiones posteriores.

La siguiente tabla describe los cambios de comportamiento controlados por cada ajuste del nivel de compatibilidad. La etiqueta `compat`*NN* se usa como un atajo para ajustar el nivel de compatibilidad *NN* usando uno de los siguientes mecanismos. Para versiones anteriores a bash-5.0, el nivel de compatibilidad puede asignarse usando la correspondiente opción shopt `compat`*NN*. Para bash-4.3 y posteriores se prefiere la variable `BASH_COMPAT`, y es requerida para bash-5.1 y versiones posteriores.

`compat31`

- entrecomillar el rhs del operador de coincidencia de expresiones regulares (=~) de la instrucción `[[` no tiene ningún efecto especial

`compat32`

- interrumpir una lista de instrucciones como `"a ; b ; c"` produce la ejecución de la siguiente instrucción en la lista (en bash-4.0 y versiones superiores, el intérprete actúa como si hubiera recibido la interrupcón, así que insterrumpir una instrucción en una lista aborta la ejecución de toda la lista)

`compat40`

- los operadores '`<`' y '`>`' de la instrucción `[[` no consideran la configuración regional actual al comparar cadenas; usan un orden de ASCII. Las versiones anteriores a bash-4.1 usan la ordenación ASCII y strcmp(3); bash-4.1 y posteriores usan la secuencia y strcoll(3) de ordenación de la actual configuración regional.

`compat41`

- en el modo posix, `time` puede estar seguido de opciones y aún será reconocido como palabra reservada (esta es la interpretación de POSIX 267)
- en modo posix, el analizador requiere que un número impar de comillas simples ocurra en la porción de *palabra* de un parámetro ${...} entre comillas dobles y los trata de forma especial, de forma que los caracteres entre las comillas simples se consideren entrecomillados (esta es la interpretación 221 de POSIX)

`compat42`

- la cadena de reemplazo en una sustitución de patrón de comillas dobles no experimenta la eliminación de comillas, como lo hacía en versiones posteriores a bash-4.2

- en el modo posix, las comillas simples son consideradas especiales al expandir la porción de *palabra* de una expansión de parámetro ${...} parámetro y pueden usarse para entrecomillar una llave de cierre u otro carácter especial (esto es parte de la interpretación 221 de 221); en versiones posteriores, las comillas simples no son especiales dentro de las expansiones de palabra de comillas dobles

`compat43`

- el intérprete no muestra un mensaje de advertencia si se realiza un intento de usar una asignación compuesta entrecomillada como un argumento para declarar (declare -a foo='(1 2)'). Las versiones posteriores advierten de que este uso está obsoleto
- los errores de expansión de palabra se consideran errores no fatales que hacen que la instrucción actual falle, incluso en el modo posix (el comportamiento predeterminado es volverlos errores fatales que provocan que el intérprete finalice)
- al ejecutar una función del intérprete, el estado del bucle (while/until/etc) no se restablece, de forma que `break` o `continue` en esa función romperá o continuará bucles en el contexto de la llamada. Bash-4.4 y posteriores restablecen el estado del bucle para evitar esto

`compat44`

- el intérprete prepara los valores usados por `BASH_ARGV` y `BASH_ARGC` para que pueden expandirse a los parámetros posicionales del intérprete incluso si no está habilitado el modo de depuración extendido
- un subintérprete hereda bucles del contexto de su padre, de forma que `break` o `continue` hará que el subintérprete finalice. Bash-5.0 y posteriores restablecen el estado del bucle para evitar la salida
- asignaciones de variable que preceden a instrucciones integradas como `export` y `readonly` que asignan atributos siguen afectando a variables con el mismo nombre en el entorno de llamada incluso si el intérprete no está en modo posix

`compat50 (asignada usando BASH_COMPAT)`

- Bash-5.1 cambió la forma en que `$RANDOM` se genera para introducir ligeramente más aleatoriedad. Si el nivel de compatibilidad del intérprete está establecido como 50 o inferior, vuelve a usar el método de bash-5.0 y versiones anteriores, así que usar una semilla para el generador de números aleatorios asignándole un valor a `RANDOM` producirá la misma secuencia que en bash-5.0
- Si la tabla hash de instrucciones está vacía, las versiones de Bash anteriores a bash-5.1 mostraban un mensaje informativo en ese sentido, incluso al producir salida que puede reutilizarse como entrada. Bash-5.1 suprime ese mensaje cuando se proporciona la opción `-l`.

7 Control de Tareas

Este capítulo discute qué es el control de tareas, cómo funciona y cómo Bash le permite acceder a sus herramientas.

7.1 Fundamentos del Control de Tareas

El control de tareas se refiere a la habilidad de parar selectivamente (suspender) la ejecución de procesos y continuar (reanudar) su ejecución en un momento posterior. Un usuario emplea típicamente esta herramienta mediante una interfaz interactiva proporcionada conjuntamente por el controlador de terminal del núcleo del sistema operativo y Bash.

El intérprete asocia una *tarea* con cada tubería. Mantiene una tabla de tareas ejecutándose actualmente, que puede ser listada con la instrucción `jobs`. Cuando Bash inicia una tarea asíncronamente, imprime una línea con este aspecto:

```
[1] 25647
```

indicando que esta tarea es el número de tarea 1 y que el ID de proceso del último proceso en la tubería asociado con esta tarea es el 25647. Cada uno de los procesos en una única tubería son miembros de la misma tarea. Bash usa la abstracción *tarea* como la base para el control de tareas.

Para facilitar la implementación de la interfaz de usuario para el control de tareas, el sistema operativo mantiene la noción de un ID de grupo de proceso actual de terminal. Los miembros de este grupo de proceso (procesos cuyo ID de grupo de proceso es igual al ID de grupo de proceso actual de terminal) reciben señales generadas por el teclado como `SIGINT`. Se dice que estos procesos están en primer plano. Los procesos en segundo plano son aquellos cuyo ID de grupo de proceso difiere del de la terminal; estos procesos son inmunes a señales generadas con el teclado. Solo se permite a los procesos en primer plano leer de o, si el usuario especifica así con `stty tostop`, escribir a la terminal. A los procesos en segundo plano que tratan de leer de (escribir a cuando está en efecto `stty tostop`) la terminal se les envía una señal `SIGTTIN` (`SIGTTOU`) por el controlador de la terminal del núcleo, que, a no ser que se atrape, suspende el proceso.

Si el sistema operativo en que Bash se está ejecutando soporta el control de tareas, Bash contiene herramientas para usarlo. Teclear el carácter *suspend* (típicamente '`^Z`', Control-Z) mientras está corriendo un proceso hace que el proceso se pare cuando trata de leer la entrada de la terminal y devuelva el control a Bash. El usuario entonces manipula el estado de esta tarea, usando la instrucción `bg` para continuarla en segundo plano, la instrucción `fg` para continuarla en primer plano o la instrucción `kill` para matarla. Un '`^Z`' tiene efecto inmediatamente, y tiene el efecto secundario adicional de hacer que la salida pendiente y las pulsaciones adicionales sean descartadas.

Hay varias formas de hacer referencia a una tarea en el intérprete. El carácter '`%`' introduce una especificación de tarea (*jobspec*).

Se puede hacer referencia al número de tarea `n` con '`%n`'. Los símbolos '`%%`' y '`%+`' hacen referencia a la noción del intérprete de la tarea actual, que es la última tarea detenida mientras estaba en primer plano o comenzada en segundo plano. Un único '`%`' (sin una especificación de tarea que lo acompañe) también se refiere a la tarea actual. Se puede hacer referencia a la tarea previa usando '`%-`'. Si hay solo una única tarea, ambos '`%+`' y '`%-`'

pueden ser usados para hacer referencia a esa tarea. En salida referente a tareas (p. ej., la salida de la instrucción `jobs`), la tarea actual siempre se marca con un '`+`', y la tarea previa con un '`-`'.

También se puede hacer referencia a una tarea usando un prefijo del nombre usado para comenzarla o usando una subcadena que aparece en la línea de órdenes. Por ejemplo, '`%ce`' hace referencia a tareas detenidas cuya instrucción comienza por '`ce`'. Usar '`%?ce`', por otro lado, hace referencia a cualquier tarea que contenga la cadena '`ce`' en la línea de instrucciones. Si el prefijo o subcadena coincide con más de una tarea, Bash informa de un error.

Simplemente nombrar una tarea puede hacerse para traerla a primer plano: '`%1`' es un sinónimo de '`fg %1`', llevando la tarea 1 desde el segundo plano a primer plano. Similarmente, '`%1 &`' reanuda la tarea 1 en segundo plano, equivalente a '`bg %1`'

El intérprete se entera inmediatamente de cuándo un tarea cambia de estado. Normalmente, Bash espera hasta que está a punto de imprimir un prompt antes de informar de cambios en un estado de tarea para no interrumpir cualquier otra salida. Si está habilitada la opción `-b` de la instrucción integrada `set`, Bash importa tales cambios inmediatamente (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68). Cualquier trap en `SIGCHLD` se ejecuta para cada proceso hijo que finaliza.

Si se realiza un intento de salir de Bash mientras las tareas están detenidas (o corriendo, si la opción `checkjobs` está habilitada —vea Sección 4.3.2 [La Instrucción Integrada Shopt], página 73—), el intérprete imprime un mensaje de advertencia, y si la opción `checkjobs` está habilitada, lista las tareas y sus estados. La instrucción `jobs` puede ser usada entonces para inspeccionar sus estados. Si se realiza un segundo intento de salir sin una instrucción que intervenga, Bash no imprime otra advertencia, y se finalizan todas las tareas detenidas.

Cuando el intérprete está esperando a una tarea o proceso usando la instrucción integrada `wait` y está activado el control de tareas, `wait` retornará cuando la tarea cambie de estado. La opción `-f` hace que `wait` espere hasta que la tarea o el proceso termine antes de retornar.

7.2 Instrucciones Integradas de Control de Tareas

`bg`

`bg [`*`espec-tarea ...`*`]`

Reanuda cada tarea *espec-tarea* suspendida en segundo plano, como si hubiera sido iniciada con '`&`'. Si no se proporciona *espec-tarea*, se usa la tarea actual. El estado de retorno es cero a no ser que se ejecute cuando el control de tareas no esté activado, o, cuando se ejecute con el control de tareas activado, la *espec-tarea* no fue encontrada o especifica una tarea que fue iniciada sin control de tareas.

`fg`

`fg [`*`espec-tarea`*`]`

Reanuda la tarea *espec-tarea* en primer plano y la hace la tarea actual. Si no se proporciona *espec-tarea*, se usa la tarea actual. El estado de retorno es el de la instrucción ubicada en primer plano, o distinto de cero si se ejecuta cuando el control de tareas está desactivado, o, cuando el control de tareas está activado,

la *espec-tarea* no especifica una tarea válida o *espec-tarea* especifica una tarea que fue iniciada sin control de tareas.

`jobs`

```
jobs [-lnprs] [espec-tarea]
jobs -x instrucciones [argumentos]
```

La primera forma lista las tareas activas. Las opciones tienen los siguientes significados:

`-l` Lista IDs de proceso además de la información normal.

`-n` Muestra información solo sobre tareas que han cambiado el estado desde que usuario fue notificado por última vez de su estado.

`-p` Lista solo el ID de proceso del líder de grupo de proceso de la tarea.

`-r` Muestra solo las tareas corriendo.

`-s` Muestra solo las tareas detenidas.

Si se pasa *espec-tarea*, la salida es restringida a información sobre esa tarea. Si no se proporciona *espec-tarea*, se lista el estado de todas las tareas.

Si se proporciona la opción `-x`, `jobs` reemplaza cualquier *espec-tarea* encontrada en *instrucción* o *argumentos* con el ID de proceso de grupo correspondiente, y ejecuta *instrucción*, pasándole *argumentos*, devolviendo su estado de salida.

`kill`

```
kill [-s especseñ] [-n numseñ] [-especseñ] espectarea or pid
kill -l|-L [estado_salida]
```

Envía una señal especificada por *especseñ* o *numseñ* al proceso llamado por la especificación de tarea *jobspec* o ID de proceso *pid*. *especseñ* es o un nombre de señal independiente de mayúsculas y minúsculas como `SIGINT` (con o sin el prefijo `SIG`) o un número de señal; *numseñ* es un número de señal. Si no están presentes ni *especseñ* ni *numseñ*, se usa `SIGTERM`. La opción `-l` lista los nombres de señal. Si se proporcionan argumentos cuando se pasa `-l`, se listan los nombres de las señales correspondientes a los argumentos, y el estado de retorno es cero. *estado_salida* es el número que especifica un número de señal o el estado de salida de un proceso terminado por una señal. La opción `-L` es equivalente a `-l`. El estado de retorno es cero si al menos una señal fue enviada exitosamente, o distinto de cero si ocurre un error o se encuentra una opción inválida.

`wait`

```
wait [-fn] [-p nombre de variable] [espec-tarea or pid ...]
```

Espera hasta que el proceso hijo especificado por cada ID, *pid* o especificación de tarea *espec-tarea* finalice y devuelva el estado de salida de la última instrucción a la que ha esperado. Si se proporciona la especificación de una tarea, se espera a todos los procesos en la tarea. Si no se proporcionan argumentos, `wait` espera a todos las tareas ejecutándose en segundo plano y a la sustitución de los últimos procesos ejecutados, si su id de proceso es igual a *$!* y el estado de retorno es

cero. Si se proporciona la opción `-n`, `wait` espera a una sola tarea de la lista de *pids* or *espec-tarea*, o si no se proporcionan argumentos, cualquier tarea para completar y devolver su estado de error. Si ninguno de los argumentos proporcionados es un hijo del intérprete, o si no se proporcion an argumentos y el intérprete no tiene ningún hijo al que no haya esperado, el estado de salida es 127. Si se proporciona la opción `-p`, el proceso o identificador de proceso de la tarea para la que el estado devuelto se asigna a la variable *nombre-var* con el argumento de opción. La variable estará inicialmente sin asignar. Esto es solo útil cuando se proporciona la opción número `-n`. Proporcionar la opción `-f` cuando está activado el control de tareas fuerza a `wait` a esperar a cada *pid* o *espec-tar* para terminar antes de devolver su estado, en vez de retornar cuando cambio de estado. Si ni *espec-tarea* ni *pids* especifica un proceso activo del intérprete, el estado de retorno es 127.

`disown`

```
disown [-ar] [-h] [espectarea ... | pid ... ]
```

Sin opciones, elimina cada *espectarea* de la tabla de tareas activas. Si se pasa la opción `-h`, no se elimina la tarea de la tabla, pero se marca de forma que no se envíe `SIGHUB` a la tarea si el intérprete recibe una `SIGHUP`. Si no está presente *espectarea* y no se proporcionan ni la opción `-a` ni la `-r`, se usa la tarea actual. Si no se proporciona ninguna *espectarea*, la opción `-a` significa eliminar o marcar todas las tareas; la opción `-r` sin un argumento *espectarea* restringe la operación a tareas en ejecución.

`suspend`

```
suspend [-f]
```

Suspende la ejecución de este intérprete hasta que recibe una señal `SIGCONT`. No se puede suspender un intérprete de acceso; la opción `-f` puede usarse para sobrescribir esto y forzar la suspensión.

Cuando no está activo el control de tareas, las instrucciones integradas `kill` y `wait` no aceptan argumentos *espectarea*. Deben ser IDs de procesos proporcionados.

7.3 Variables de Control de Tareas

`auto_resume`

Esta variable controla cómo el intérprete interactúa con el usuario y el control de tareas. Si esta variable existe, las instrucciones simples de una palabra sin redirección son tratadas como candidatos para la reanudación de una tarea existente. No se permite ambigedad; si hay más de una tarea que comienza por la cadena tecleada, se seleccionará la tarea más recientemente accedida. El nombre de una tarea detenida, en este contexto, es la línea de orden usada para ejecutarla. Si esta variable está establecida al valor '`exact`', la cadena proporcionada debe coincidir con el nombre de una tarea detenida recientemente; si establecida a '`substring`', la cadena proporcionada necesita coincidir con una subcadena del nombre de una tarea detenida. El valor '`substring`' proporciona funcionalidad análoga al ID de tarea '`%?`' (véase Sección 7.1 [Fundamentos del

Control de Tareas], página 118). Si está asignada con otro valor, la cadena proporcionada debe ser un prefijo de un nombre de tarea detenida; esto proporciona funcionalidad análoga al ID de tarea '%'.

8 Edición en Línea de Órdenes

Este capítulo describe las funcionalidades básicas de la interfaz de edición en línea de órdenes de GNU. La edición de línea de órdenes está proporcionada por la biblioteca Readline, que es usada por varios programas diferentes, incluido Bash. La edición de línea de órdenes es habilitada por defecto cuando se usa un intérprete interactivo, a no ser que se proporcione la opción `--noediting` en la llamada al intérprete. La edición de línea también se usa al usar la opción `-e` para la instrucción integrada `read` (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56). Por defecto, las órdenes de edición de línea son similares a las de Emacs. También está disponible una edición de línea estilo vi. La edición de línea puede ser habilitada en cualquier momento usando las opciones `-o emacs` o `-o vi` para la instrucción integrada `set` (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68) o deshabilitada usando las opciones `+o emacs` o `+o vi` para `set`.

8.1 Introducción a la Edición de Línea

Los siguientes párrafos describen la notación usada para representar pulsaciones de teclas.

El texto *`C-k`* se lee como 'Control-K' y describe el carácter producido cuando se pulsa la tecla `k` mientras se oprime la tecla Control.

El texto *`M-k`* se lee como 'Meta-K' y describe el carácter producido cuando la tecla Meta (si tiene una) se oprime y se pulsa la tecla `k`. La tecla Meta está etiquetada como `ALT` en muchos teclados. En teclados con dos teclas etiquetadas como `ALT` (normalmente a ambos lados de la barra espaciadora), el `ALT` en el lado izquierdo está generalmente establecido para funcionar como tecla Meta. La tecla `ALT` en el derecho también puede ser configurado para funcionar como una tecla Meta o puede ser configurado como algún otro modificador, como la tecla Compose para escribir caracteres acentuados.

Si no tiene un tecla Meta o `ALT`, u otra tecla funcionando como una tecla Meta, se puede generar la misma pulsación escribiendo `ESC` *primero* y después escribiendo `k`. Este proceso se conoce como *metaficar* la tecla `k`.

El texto *`M-C-k`* se lee como 'Meta-Control-k' y describe el carácter producido al *metaficar* *`C-k`*.

Además, varias teclas tienen sus propios nombres. Específicamente, `SUPR`, `ESC`, `LFD`, `SPC`, `ENT` y `TAB` todas se representan a sí mismas al ser vistas en este texto o en un archivo de inicialización (véase Sección 8.3 [Archivo de Inicialización de Readline], página 126). Su su teclado carece de una tecla `LFD`, teclear `C-j` producirá el carácter deseado. La tecla `ENT` puede estar etiquetada como `Retorno` o `Entrar` en algunos teclados.

8.2 Interacción con Readline

Normalmente durante una sesión interactiva escribe una larga línea de texto, para darse cuenta de que la primera palabra en la línea está mal escrita. La biblioteca Readline le proporciona un conjunto de órdenes para manipular el texto a medida que lo escribe, permitiéndole solo arreglar su errata, y no forzarle a volver a escribir la mayor parte de la línea. Usando estas órdenes de edición, mueve el cursor al lugar que necesita corrección y elimina o inserta el texto de las correcciones. Entonces, cuando está satisfecho con la línea, simplemente pulsa `ENT`. No tiene que estar al final de la línea para pulsar `ENT`; se acepta la línea completa independientemente de la ubicación del cursor dentro de la línea.

8.2.1 Mínimos Esenciales de Readline

Para introducir caracteres en la línea, simplemente escríbalos. El carácter escrito aparece donde estaba el cursor, y entonces el cursor se mueve un carácter a la derecha. Si escribió mal un carácter, puede usar su carácter de borrado para volver atrás y borrar el carácter mal escrito.

Algunas veces puede escribir mal un carácter, y no darse cuenta del error hasta haber escrito varios otros caracteres. En ese caso, puede pulsar *C-b* para mover el cursor a la izquierda y, después, corregir su fallo. A continuación, puede mover el cursor a la derecha con *C-f*.

Cuando añade texto en medio de una línea, se dará cuenta de que los caracteres a la derecha del cursor son 'empujados' dejar espacio al texto que ha insertado. Del mismo modo, cuando cuando borra texto antes del cursor, los caracteres a la derecha del cursor son 'arrastrados atrás' para llenar el espacio en blanco creado por la eliminación del texto. A continuación, una lista de lo mínimo necesario para editar el texto de una línea de entrada.

C-b Se mueve atrás un carácter.

C-f Se mueve un carácter adelante.

`SUPR` or `Retroceso`
Borra el carácter a la izquierda del cursor.

C-d Borra el carácter bajo el cursor.

Imprimir caracteres
Inserta el carácter en la línea en la posición del cursor.

C-_ o *C-x C-u*
Deshace la última orden de edición. Puede deshacer hacia atrás hasta una línea vacía.

(Dependiendo de su configuración, la tecla `Retroceso` está establecida para eliminar el carácter a la izquierda del cursor y la tecla `DEL` establecida para eliminar el carácter bajo el cursor, como *C-d*, en vez del carácter a la izquierda del cursor.)

8.2.2 Órdenes de Movimiento de Readline

La tabla de arriba describe las pulsaciones de teclas más básicas que necesita para realizar ediciones de la línea de entrada. Para su conveniencia, se han añadido muchas otras órdenes además de *C-b*, *C-f*, *C-d* y `SUPR`. He aquí algunas de las órdenes para moverse más rápido por la línea.

C-a Se mueve al comienzo de la línea.

C-e Se mueve al final de la línea.

M-f Se mueve adelante una palabra, donde una palabra está compuesta de letras y dígitos.

M-b Se mueve atrás una palabra.

C-l Limpia la pantalla, volviendo a imprimir la línea actual en lo más alto.

Observe como *C-f* se mueve adelante un carácter, mientras que *M-f* se mueve adelante una palabra. Es una convención informal que las pulsaciones de control operen en caracteres y las pulsaciones meta en palabras.

8.2.3 Órdenes de Cortar de Readline

Cortar texto significa eliminar el texto de la línea, pero guardarlo seguidamente para un uso posterior, normalmente *pegándolo* (reinsertándolo) en la línea. (En inglés se dice 'kill' para cortar y 'yank' para pegar, pero 'copy' y 'paste' es una jerga más reciente.)

Si la descripción de una instrucción dice que 'corta' texto, puede estar seguro de que puede volver a obtener el texto en un lugar diferente (o el mismo) después.

Cuando usa una instrucción de cortar, el texto se guarda en un *kill-ring* [anillo de la muerte, literalmente, nosotros lo llamaremos anillo de corte]. Un número cualquiera de cortes consecutivos guarda todo el texto cortado junto, para que cuando lo pegue de nuevo lo obtenga todo. El anillo de corte no es específico para la línea; el texto que cortó en una línea escrita previamente está disponible para ser pegado después, cuando está escribiendo otra línea.

He aquí la lista de órdenes para cortar texto.

`C-k` Corta el texto desde la posición del cursor actual hasta el final de la línea.

`M-d` Corta desde el cursor hasta el final de la palabra actual o, si está entre palabras, hasta el final de la siguiente palabra. Los límites de palabra son los mismos usados por `M-f`.

`M-SUPR` Corta desde el cursor el principio de la palabra actual o, si está entre palabras, hasta el principio de la palabra anterior. Los límites de palabra son los mismos usados por `M-b`.

`C-w` Corta desde el cursor hasta el espacio en blanco anterior. Esto es diferente a `M-SUPR` porque difieren los límites de palabra.

He aquí cómo *pegar* el texto de nuevo en la línea. Pegar significa copiar el texto más recientemente cortado del búfer de corte.

`C-y` Pega el texto más recientemente cortado de nuevo en el búfer en el cursor.

`M-y` Rota el kill-ring, y pega la nueva parte superior. Solo puede hacer esto si la anterior orden es `C-y` o `M-y`.

8.2.4 Argumentos de Readline

Puede pasar argumentos numéricos a órdenes de Readline. Algunas veces el argumento actúa como una cuenta de repetición, otras veces es el *signo* del argumento lo importante. Si pasa un argumento negativo a una orden que normalmente actúa en una dirección hacia delante, esa orden actuará en una dirección hacia atrás. Por ejemplo, para cortar texto atrás hasta el principio de la línea, puede escribir '`M-- c-k`'.

La manera general de pasar argumentos numéricos a una orden es escribir dígitos meta antes de la orden. Si el primer 'dígito' escrito es un signo de menos ('`-`'), entonces el signo del argumento será negativo. Una vez que haya escrito más de un dígito meta para empezar el argumento, puede escribir el resto de dígitos y después la instrucción. Por ejemplo, para darle a la orden `C-d` un argumento de 10, podría escribir '`M-1 0 C-d`', lo que eliminará los siguientes diez caracteres en la línea de entrada.

8.2.5 Buscando Instrucciones en el Historial

Readline proporciona órdenes para buscar a través del historial de instrucciones (véase Sección 9.1 [Servicios del Historial de Bash], página 161) para líneas que contienen una cadena especificada. Hay dos modos de búsqueda: *incremental* y *no incremental*.

Las búsquedas incrementales comienzan antes de que el usuario haya terminado de escribir la cadena de búsqueda. Al escribir cada carácter de la cadena de búsqueda, Readline muestra la siguiente entrada del historial que coincide con la cadena escrita hasta entonces. Una búsqueda incremental requiere solo tantos caracteres como sean necesarios para encontrar la entrada del historial deseada. Para buscar hacia tras en el historial una cadena concreta, escriba `C-r`. Escribir `C-s` busca hacia delante a través del historial. Los caracteres presentes en el valor de la variable `isearch-terminators` se usan para terminar una búsqueda incremental. Si no se ha asignado un valor a esa variable, los caracteres `ESC` y `C-J` terminarán una búsqueda incremental. `C-g` abortará una búsqueda incremental y restaurará la línea original. Cuando la búsqueda se termina, la entrada del historial que contiene la cadena de búsqueda se convierte en la línea actual.

Para encontrar otras entradas coincidentes en la lista del historial, escriba `C-r` o `C-s` según sea apropiado. Esto buscará atrás o adelante en el historial la siguiente entrada que coincida con la cadena de búsqueda escrita hasta ahora. Cualquier otra secuencia de teclas asociada a una orden Readline terminará la búsqueda y ejecutará esa orden. Por ejemplo, un `RET` terminará la búsqueda y ejecutará la línea, ejecutándo así la instrucción de la lista del historial. Una orden de movimiento terminará la búsqueda, convertirá la última línea encontrada en la línea actual y comenzará a editar.

Readline recuerda la última cadena de la búsqueda incremental. Si se teclean dos `C-r`s sin caracteres que intervienen definiendo una nueva cadena de búsqueda, se usa cualquier cadena de búsqueda recordada.

Las búsquedas no incrementales leen la cadena de búsqueda entera antes de empezar a buscar líneas de historial coincidentes. La cadena de búsqueda puede ser escrita por el usuario o como parte de los contenidos de la línea actual.

8.3 Archivo de Inicialización de Readline

Aunque la librería de Readline viene con un conjunto de atajos de teclas estilo Emacs instalado por defecto, es posible utilizar un conjunto diferente de atajos de teclas. Cualquier usuario puede personalizar los programas que usa Readline poniendo instrucciones en un archivo *inputrc*, de forma convencional en su directorio hogar. El nombre de este archivo se toma del valor de la variable del intérprete `INPUTRC`. Si esa variable no está establecida, el predeterminado es `~/.inputrc`. Si ese archivo no existe o no se puede leer, el predeterminado final es `/etc/inputrc`.

Cuando se inicia un programa que usa la librería Readline, es leído el archivo de inicialización y se establecen las asociaciones de teclas.

Además, la orden `C-x C-r` vuelve a leer este archivo de inicialización, incorporando así todos los cambios que hayas podido hacerle.

8.3.1 Sintaxis del Archivo de Inicialización de Readline

Solo hay unas pocas construcciones básicas permitidas en el archivo de inicialización de Readline. Las líneas en blanco son ignoradas. Las líneas que comienzan por '`#`' son co-

mentarios. Las líneas que comienzan por '`$`' indican construcciones condicionales (véase Sección 8.3.2 [Construcciones Condicionales de Inicialización], página 135). Otras líneas denotan ajustes de variables y asociaciones de teclas.

Ajustes de Variables

Puede modificar el comportamiento en ejecución de Readline alterando los valores de variables en Readline usando la orden `set` dentro del archivo de inicialización. La sintaxis es simple:

```
set variable valor
```

He aquí, por ejemplo, cómo cambiar del modo predeterminado de asociaciones de teclas estilo Emacs para usar las órdenes de edición de línea estilo `vi`:

```
set editing-mode vi
```

Los nombres y valores de variables, donde sea apropiado, se reconocen sin importar mayúsculas y minúsculas. Los nombres de variable no reconocidos son ignorados.

Las variables booleanas (aquellas que pueden establecerse en on u off) se establecen a on si el valor es nulo o vacío, *on* (independiente de mayúsculas y minúsculas) o 1. Cualquier otro valor hace que la variable sea asignada a off.

La instrucción `bind -V` lista los nombres y valores de variable actuales de Readline. Véase Sección 4.2 [Instrucciones Integradas de Bash], página 56.

Una gran parte del comportamiento de ejecución se puede cambiar con las siguientes variables.

`bell-style`
: Controla qué sucede cuando Readline quiere sonar el timbre de la terminal. Si está puesta en '`none`', Readline nunca suena el timbre. Si está puesta en '`visible`', Readline usa un timbre visible si hay uno disponible. Si está puesta en '`audible`' (por defecto), Readline trata de sonar el timbre de la terminal.

`bind-tty-special-chars`
: Si está puesta en '`on`' (por defecto), Readline trata de asociar los caracteres de control tratados de forma especial por el controlador de la terminal del núcleo a sus equivalentes de Readline.

`blink-matching-paren`
: Si está puesta en '`on`', Readline trata de mover el cursor brevemente a un paréntesis de apertura cuando se inserta un paréntesis de cierre. Por defecto es '`off`'.

`colored-completion-prefix`
: Si está puesta en '`on`', al listar compleciones, Readline muestra el prefijo común del conjunto de compleciones posibles usando un color diferente. Las definiciones de color son tomadas del valor de la variable de entorno `LS_COLORS`. Por defecto es '`off`'.

`colored-stats`
: Si está puesta en '`on`', Readline muestra las posibles compleciones usando diferentes colores para indicar su tipo de archivo. Las

definiciones de color se toman del valor de la variable de entorno `LS_COLORS`. Por defecto es '`off`'.

`comment-begin`
: La cadena que se inserta al principio de la línea cuando se ejecuta la instrucción `insert-comment`. El valor predeterminado es `"#"`.

`completion-display-width`
: El número de columnas de pantalla usadas para mostrar posibles coincidencias al realizar la compleción. El valor es ignorado si es menor que 0 o mayor que la anchura de la pantalla de la terminal. Un valor de 0 hará que se muestre una coincidencia por línea. El valor predeterminado es -1.

`completion-ignore-case`
: Si está puesta en '`on`', Readline realiza la coincidencia de nombre de archivo y compleción sin importar mayúsculas y minúsculas. El valor predeterminado es '`off`'.

`completion-map-case`
: Si está puesta en '`on`' y está habilitada *completion-ignore-case*, Readline trata los guiones ('`-`') y barras bajas ('`_`') como equivalentes al realizar coincidencia y compleción de nombre de archivo independientemente de mayúsculas y minúsculas. El valor predeterminado es '`off`'.

`completion-prefix-display-length`
: La longitud en caracteres del prefijo común de una lista de posibles compleciones que se muestra sin modificación. Al establecerse a un valor mayor que cero, los prefijos comunes más largos que este valor son reemplazados con una elipsis al mostrar posibles compleciones.

`completion-query-items`
: El número de posibles compleciones que determina cuando el usuario es preguntado si la lista de posibilidades debería ser mostrada. Si el número de posibles compleciones es más grande que este valor, Readline preguntará al usuario si desea o no verla; de lo contrario, son simplemente listadas. A esta variable se le debe asignar un valor entero mayor o igual a 0. Un valor negativo significa que Readline nunca debería preguntar. El límite predeterminado es `100`.

`convert-meta`
: Si está puesta en '`on`', Readline convertirá los caracteres con el octavo bit asignado a una secuencia de teclas ASCII quitando el octavo bit y prefijando un carácter `ESC`, convirtiéndolos en una secuencia de teclas prefijadas por meta. El valor predeterminado es '`on`', pero será puesto en '`off`' si la configuración regional es una que contiene caracteres de ocho bits.

`disable-completion`
: Puesta en '`On`', Readline inhibirá la compleción de palabra. Los caracteres de compleción serán insertados en la línea como si hubieran sido asociados a `self-insert`. Lo predeterminado es '`off`'.

`echo-control-characters`
: Si está puesta en '`on`', en sistemas operativos que indican que lo soportan, readline muestra un carácter correspondiente a una señal generada desde el teclado. Lo predeterminado es '`on`'.

`editing-mode`
: La variable `editing-mode` controla que conjunto predeterminado de asociaciones de teclado es usado. Por defeco, Readline se inicia en el modo de edición de Emacs, donde las pulsaciones de teclado son lo más similar a Emacs. Esta variable puede establecerse tanto a '`emacs`' como a '`vi`'.

`emacs-mode-string`
: Si la variable *show-mode-in-prompt* está habilitada, esta cadena se muestra inmediatamente antes de la última línea del prompt primario cuando está activado el modo de edición de emacs. El valor se expande como una asociación de teclas, de forma que está disponible el conjunto estándar de prefijos meta- y de control y secuencias de barras invertidas de escape. Use los escapes '`\1`' y '`\2`' para empezar y terminar secuencias de caracteres no imprimibles, que pueden ser usados para encerrar una secuencia de control de terminal en la cadena de modo. Lo predeterminado es '`@`'.

`enable-bracketed-paste`
: Cuando está puesta en '`On`', Readline configurará la terminal de forma que la activará para insertar cada pegada en el búfer de edición como una cadena simple de caracteres, en vez de tratar cada carácter como si hubiera sido leído del teclado. Esto puede evitar que caracteres pegados sean interpretados como ordenes de edición. Lo predeterminado es '`off`'.

`enable-keypad`
: Cuando está puesta en '`on`', Readline tratará de habilitar el teclado numérico al ser llamado. Algunos sistemas necesitan esto para activar las teclas de flechas. Lo predeterminado es '`off`'.

`enable-meta-key`
: Cuando está puesta en '`on`', Readline tratará de activar cualquier tecla modificadora meta que la terminal afirme soportar cuando sea llamada. En muchas terminales, la tecla meta se usa para enviar caracteres de ocho bits. Lo predeterminado es '`on`'.

`expand-tilde`
: Si está puesta en '`on`' se realiza la expansión de virgulilla cuando Readline intenta la compleción de palabra. Por defecto es '`off`'.

`history-preserve-point`
: Si está puesta en '`on`', el código del historial trata de ubicar el punto (la posición del cursor actual) en la misma ubicación en cada línea del historial recuperada con `previous-history` o `next-history`. Por defecto es '`off`'.

`history-size`
: Establece el número máximo de entradas del historial guardadas en la lista del historial. Si está puesta en cero, todas las entradas del historial existentes son eliminadas y no se guarda ninguna nueva entrada. Si se le asigna un valor menor que cero, el número de entradas del historial no está limitado. Por defecto, el número de entradas del historial no está limitado. Si se realiza un intento para establecer *history-size* a un valor no numérico, el máximo número de entradas del historial será establecido en 500.

`horizontal-scroll-mode`
: Esta variable se puede poner en '`on`' u '`off`'. Ponerla en '`on`' significa que el texto de las líneas que está siendo editado se desplazará horizontalmente en una única línea de la pantalla cuando estas no sean más largas que la altura de la pantalla, en vez de envolverla en una nueva línea de la pantalla. Por defecto, esta variable está puesta en '`off`'.

`input-meta`
: Si está puesta en '`on`', Readline habilitará entrada de ocho bits (no eliminará el octavo bit en los caracteres que lea), sin importar lo que la terminal afirme soportar. El valor predeterminado es '`off`', pero Readline lo establecerá en '`on`' si la configuración regional contiene caracteres de ocho bits. El nombre `meta-flag` es un sinónimo para esta variable.

`isearch-terminators`
: La cadena de caracteres que debería terminar una búsqueda incremental sin ejecutar posteriormente el carácter como una instrucción (véase Sección 8.2.5 [Búsqueda], página 126). Si a esta variable no se le ha dado un valor, los caracteres `ESC` y *`C-J`* terminarán una búsqueda incremental.

`keymap`
: Ajusta la idea de Readline del mapa de teclas actual para las órdenes de asociación de teclas. Nombres integrados de `mapas de teclas` son `emacs`, `emacs-standard`, `emacs-meta`, `emacs-ctlx`, `vi`, `vi-move`, `vi-command` y `vi-insert`. `vi` es equivalente a `vi-command` (`vi-move` es también un sinónimo); `emacs` es equivalente a `emacs-standard`. Las aplicaciones pueden añadir nombres adicionales. El valor predeterminado es `emacs`. El valor de la variable `editing-mode` también afecta al mapa de teclas predeterminado.

`keyseq-timeout`
: Especifica la duración que Readline esperará un carácter al leer una secuencia de teclas ambigua (una que puede formar una secuencia de teclas completa usando la entrada leída hasta entonces o pueda tomar una entrada adicional para completar una secuencia de teclas más larga). Si no se recibe entrada dentro del límite de tiempo, Readline usará la más corta pero completa secuencia de teclas. Readline usa este valor para determinar si la entrada está disponible o no en la actual fuente de entrada (`rl_instream` por defecto). El valor es especificado en milisegundos, así que un valor de 1000 significa que Readline esperará una entrada adicional un segundo. Si esta variable se establece a un valor menor o igual a cero o a un valor no numérico, Readline esperará hasta que se pulse otra tecla para decidir qué secuencia completar. El valor predeterminado es `500`.

`mark-directories`
: Si está puesta en '`on`', se añade una barra a los nombres de directorios completados. Por defecto es '`on`'.

`mark-modified-lines`
: Esta variable, cuando está puesta en '`on`', hace que Readline muestre un asterisco ('`*`') al comienzo de las líneas del historial que hayan sido modificadas. Esta variable está en '`off`' por defecto.

`mark-symlinked-directories`
: Si está puesta en '`on`', se añade una barra a los nombres completados que son enlaces simbólicos a directorios (sujeto al valor de `mark-directories`). Por defecto es '`off`'.

`match-hidden-files`
: Esta variable, cuando es puesta en '`on`', hace que Readline coincida los archivos cuyos nombres comiencen por un '`.`' (archivos ocultos) al realizar compleción de nombre de archivo. Si está puesta en '`off`', se debe proporcionar el '`.`' inicial por el usuario en el nombre de archivo a ser completado. Esta variable está en '`on`' por defecto.

`menu-complete-display-prefix`
: Si está puesta en '`on`', la compleción de menú muestra el prefijo común de la lista de posibles compleciones (que puede ser vacía) antes de iterar sobre la lista. Por defecto es '`off`'.

`output-meta`
: Si está puesta en '`on`', Readline mostrará caracteres con el octavo bit establecido directamente en vez de como una secuencia de escape prefijada con meta. El valor predeterminado es '`off`', pero Readline lo pondrá en '`on`' si la configuración regional contiene caracteres de ocho bits.

`page-completions`
: Si está puesta en '`on`' Readline usa un paginador interno parecido a `more` para mostrar una pantalla completa de posibles compleciones cada vez. Esta variable está en '`on`' por defecto.

`print-completions-horizontally`
: Si está puesta en '`on`', Readline mostrará compleciones con coincidencias ordenadas horizontalmente en orden alfabético, en vez de pantalla abajo. Por defecto es '`off`'.

`revert-all-at-newline`
: Si está puesta en '`on`', Readline deshará todos los cambios a las líneas del historial antes de retornar cuando se ejecute `accept-line`. Por defecto, las líneas del historial pueden ser modificadas y conservar listas de deshacer entre llamadas a `readline`. Lo predeterminado es '`off`'.

`show-all-if-ambiguous`
: Esto altera el comportamiento predeterminado de las funciones de compleción. Si está puesto en '`on`', palabras que tengan más de una posible compleción hacen que las coincidencias sean listadas inmediatamente en vez de sonar el timbre. El valor predeterminado es '`off`'.

`show-all-if-unmodified`
: Esto altera el comportamiento predeterminado de las funciones de compleción en un estilo similar al de *show-all-if-ambiguous*. Si está puesto en '`on`', las palabras que tienen más de una posible compleción sin una posible compleción parcial (las posibles compleciones no comparten un prefijo común) hacen que las coincidencias sean mostradas inmediatamente en vez de sonar el timbre. El valor predeterminado es '`off`'.

`show-mode-in-prompt`
: Si está puesta en '`on`', añade una cadena al comienzo del prompt indicando el modo de edición: emacs, orden vi o inserción vi Las cadenas de modo se pueden establecer por el usuario (p. ej., *emacs-mode-string*). El valor predeterminado es '`off`'.

`skip-completed-text`
: Si está puesta en '`on`', esto altera el comportamiento predeterminado de compleción al insertar una única coincidencia en la línea. Solo está activa al realizar compleciones en mitad de palabra. Si está habilitada, readline no inserta caracteres desde la compleción que modifican caracteres después del punto en la palabra que está siendo completada, así que las partes de la palabra que siguen al cursor no se duplican. Por ejemplo, si esto está habilitado, intentar la compleción cuando el cursor está después de la '`e`' en '`Makefile`' reultará en '`Makefile`' en vez de '`Makefilefile`', asumiendo que hay una única compleción posible. El valor predeterminado es '`off`'

`vi-cmd-mode-string`
: Si la variable *show-mode-in-prompt* está habilitada, esta cadena se muestra inmediatamente antes de la última línea del prompt primario cuando está activado el modo de edición de vi y está en modo comando. El valor se expande como una asociación de teclas, de forma que está disponible el conjunto estándar de prefijos meta- y de control y secuencias de barras invertidas de escape. Use los escapes '`\1`' y '`\2`' para empezar y terminar secuencias de caracteres no imprimibles, que pueden ser usados para encerrar una secuencia de control de terminal en la cadena de modo. Lo predeterminado es '`(cmd)`'.

`vi-ins-mode-string`
: Si la variable *show-mode-in-prompt* está habilitada, esta cadena se muestra inmediatamente antes de la última línea del prompt primario cuando está activado el modo de edición de vi y está en modo de inserción. El valor se expande como una asociación de teclas, de forma que está disponible el conjunto estándar de prefijos meta- y de control y secuencias de barras invertidas de escape. Use los escapes '`\1`' y '`\2`' para empezar y terminar secuencias de caracteres no imprimibles, que pueden ser usados para encerrar una secuencia de control de terminal en la cadena de modo. Lo predeterminado es '`(ins)`'.

`visible-stats`
: Si está puesta en '`on`', se añade un carácter que denota el tipo de un archivo al nombre de archivo durante el listado de posibles compleciones. Por defecto es '`off`'.

Asociaciones de Teclas
: La sintaxis para controlar las asociaciones de teclas en el archivo de inicialización es sencilla. Primero necesita encontrar el nombre de la orden que quiere cambiar. Las siguientes secciones contienen tablas del nombre de orden, la asociación de teclado predeterminada, si hay, y una descripción corta de lo que hace la orden.

  Una vez que sepa el nombre de la orden, simplemente ubique en una línea en el archivo de inicialización el nombre de la tecla que desea asociar la orden, dos puntos y el nombre de la orden. No puede haber espacio entre el nombre de la tecla y los dos puntos, que serán interpretados como parte del nombre de la tecla. El nombre de la tecla se puede expresar de diferentes formas, dependiendo de lo que encuentre más cómodo.

  Además de nombres de órdenes, readline permite que sean asociadas teclas a una cadena que se inserta cuando la tecla se pulsa (una *macro*).

  La instrucción `bind -p` muestra los nombres de funciones y las asociaciones de Readline en un formato que se puede poner directamente en un archivo de inicialización. Véase Sección 4.2 [Instrucciones Integradas de Bash], página 56.

  nombre-de-tecla: *nombre-de-función* o *macro*
  : *nombre-de-tecla* es el nombre de una tecla enunciada en inglés. Por ejemplo:

```
Control-u: universal-argument
Meta-Rubout: backward-kill-word
Control-o: "> output"
```

En el ejemplo anterior, *C-u* está asociada a la función `universal-argument`, *M-SUPR* está asociada a la función `backward-kill-word` y *C-o* está asociada a ejecutar la macro expresada en el lado derecho (es decir, a insertar el texto '`> output`' en la línea).

Un número de nombres de caracteres simbólicos se reconocen al procesar esta sintaxis de asociación de teclas: *DEL*, *ESC*, *ESCAPE*, *LFD*, *NEWLINE*, *ENT*, *RETURN*, *RUBOUT*, *SPACE*, *SPC*, and *TAB*.

"*secteclas*": *nombre-de-función* o *macro*

secteclas difiere de *nombre-de-tecla* anterior en que las cadenas denotan una secuencia de teclas entera que puede ser especificada, ubicando la secuencia de teclas en comillas dobles. Se pueden usar algunas secuencias de escape de teclas estilo Emacs, como en el siguiente ejemplo, pero no se reconocen los nombres de caracteres especiales.

```
"\C-u": universal-argument
"\C-x\C-r": re-read-init-file
"\e[11~": "Tecla de Función 1"
```

En el ejemplo anterior, *C-u* está de nuevo asociada a la función `universal-argument` (como estaba en el primer ejemplo), '*C-x C-r*' está asociada a la función `re-read-init-file` y '`ESC [ 1 1 ~`' está asociada a insertar el texto '`Tecla de Función 1`'.

Las siguientes secuencias de escape estilo GNU Emacs están disponibles al especificar secuencias de caracteres:

`\C-`	prefijo control
`\M-`	prefijo meta
`\e`	un carácter de escape
`\\`	barra invertida
`\"`	`"`, un símbolo de comilla doble
`\'`	`'`, una comilla simple o apóstrofo

Además de las secuencias de escape estilo GNU Emacs, está disponible un segundo conjunto de secuencias de escape:

`\a`	alerta (timbre)
`\b`	retroceso
`\d`	eliminar
`\f`	salto de página

`\n` nueva línea

`\r` retorno de carro

`\t` tabulación horizontal

`\v` tabulación vertical

`\nnn` la secuencia de caracteres de ocho bits cuyo valor es el valor octal *nnn* (de uno a tres dígitos)

`\x`*HH* el carácter de ocho bits cuyo valor es el valor hexadecimal *HH* (uno o dos dígitos hexadecimales)

Al introducir el texto de una macro, las comillas simples o dobles se deben usar para indicar una definición de macro. Se asume que el texto sin entrecomillar es un nombre de función. En el cuerpo de la macro, los escapes de barra invertida descritos arriba son expandidos. La barra invertida entrecomillará cualquier otro carácter en el texto de la macro, incluyendo a '`"`' y '`'`'. Por ejemplo, la siguiente asociación hará que '`C-x \`' inserte una única '`\`' en la línea:

```
"\C-x\\": "\\"
```

8.3.2 Construcciones Condicionales de Inicialización

Readline implementa una herramienta similar en espíritu a las funcionalidades condicionales de compilación del preprocesador C que permiten que asociaciones de teclas y ajustes de variables se realicen como el resultado de comprobaciones. Hay cuatro directivas del analizador usadas.

`$if` La construcción `$if` permite que se hagan asociaciones basadas en el modo de edición, la terminal que está siendo usada o las aplicaciones que usan Readline. El texto de la comprobación, después de cualquier operador de comparación, se extiende hasta el final de la línea; a no ser que se especifique lo contrario, no se requieren caracteres para aislarlo.

`mode` La forma `modo=` de la directiva `$if` se usa para comprobar si Readline está en el modo `emacs` o `vi`. Esto puede ser usado en conjunto con la orden '`set keymap`', por ejemplo, para establecer asociaciones en los mapas de teclas `emacs-standard` y `emacs-ctlx` solo si Readline está empezando en modo `emacs`.

`term` La forma `term=` puede ser usada para incluir asociaciones de teclas específicas de la terminal, quizá para asociar la salida de secuencias de teclas por las teclas de función de la terminal. La palabra al lado izquierdo del '`=`' se comprueba sobre tanto el nombre completo de la terminal como la porción del nombre de la terminal antes del primer '`-`'. Esto permite que `sun` coincida tanto con `sun` y `sub-cmd`, por ejemplo.

`version` La prueba `version` puede usarse para realizar comparaciones entre versiones específicas de Readline. La `version` se expande a la actual versión de Readline. El conjunto de operadores de comparación incluye '`=`' (y '`==`'), '`!=`', '`<=`', '`>=`', '`<`' y '`>`'. El número de

versión proporcionado en el lado izquierdo del operador consiste en un número de versión mayor, un punto decimal opcional y una versión menor opcional (p. ej., '7.1'). Si se omite la versión menor, se asume que es '0'. El operador puede separarse de la cadena `version` y del argumento de número de versión mediante espacio en blanco. El siguiente ejemplo asigna una variable si la versión de Readline usada es la 7.0 o una más nueva:

```
$if version >= 7.0
set show-mode-in-prompt on
$endif
```

`application`
: La construcción *application* se usa para incluir ajustes específicos de aplicación. Cada programa usando la librería Readline asigna el *nombre de aplicación*, y puede comprobar por un valor particular. Esto se podría usar para asociar secuencias de teclas a funciones útiles para un programa específico. Por ejemplo, la siguiente orden añade una secuencia que entrecomilla la palabra actual o previa en Bash:

```
$if Bash
# Entrecomilla la palabra actual o previa
"\C-xq": "\eb\"\ef\""
$endif
```

`variable` La construcción *variable* proporciona simples comprobaciones de igualdad para las variables y los valores de Readline. Los operadores de comparación permitidos son '`=`', '`==`' y '`!=`'. El nombre de variable debe estar separado por el operador de comparación por espacio en blanco; el operador puede separarse del valor de la derecha con un espacio en blanco. Tanto las variables de cadena y las booleanas pueden ser comprobadas. Las variables booleanas pueden ser comprobadas con los valores *on* y *off*. El siguiente ejemplo es equivalente a la comprobación `mode=emacs` descrita anteriormente:

```
$if editing-mode == emacs
set show-mode-in-prompt on
$endif
```

`$endif` Esta instrucción, como se vio en el ejemplo anterior, termina una orden `$if`.

`$else` Las órdenes en esta rama de la directiva `$if` son ejecutadas si falla la comprobación.

`$include` Esta directiva toma un único nombre de archivo como un argumento y lee órdenes y asociaciones de ese archivo. Por ejemplo, la siguiente directiva lee de `/etc/inputrc`:

```
$include /etc/inputrc
```

8.3.3 Archivo Init de Muestra

He aquí un ejemplo de un archivo *inputrc*. Este ilustra la asociación de teclas, la asignación de variables y la sintaxis condicional.

```
# Este archivo controla el comportamiento de la edición de
# la línea de entrada para programas que usan la
# biblioteca de GNU Readline. Programas existentes
# incluyen FTP, Bash y GDB.
#
# Puede volver a leer el archivo inputrc con C-x C-r.
# Las líneas que comienzan por '#' son comentarios.
#
# Primero, incluye las asociaciones de alcance a todo el
# sistema de /etc/Inputrc
$include /etc/Inputrc

#
# Establece varias asociaciones para el modo emacs.

set editing-mode emacs

$if mode=emacs

Meta-Control-h: backward-kill-word Texto ignorado tras nombre de función

#
# Teclas de flecha en modo de teclado
#
#"\M-OD":        backward-char
#"\M-OC":        forward-char
#"\M-OA":        previous-history
#"\M-OB":        next-history
#
# Teclas de flecha en modo ANSI
#
"\M-[D":        backward-char
"\M-[C":        forward-char
"\M-[A":        previous-history
"\M-[B":        next-history
#
# Teclas de flecha en modo de teclado de 8 bits
#
#"\M-\C-OD":       backward-char
#"\M-\C-OC":       forward-char
#"\M-\C-OA":       previous-history
#"\M-\C-OB":       next-history
#
# Teclas de flechas en modo ANSI de 8 bits
#
#"\M-\C-[D":       backward-char
```

```
#"\M-\C-[C":       forward-char
#"\M-\C-[A":       previous-history
#"\M-\C-[B":       next-history

C-q: quoted-insert

$endif

# Una asociación de estilo antiguo.  Esta resulta ser la predeterminada.
TAB: complete

# Macros que son convenientes para la interacción del intérprete
$if Bash
# edita la ruta
"\C-xp": "PATH=${PATH}\e\C-e\C-a\ef\C-f"
# prepara para teclear una palabra entrecomillada --
# inserta comillas dobles de apertura y cierre
# y se mueve justo después de la comilla de apertura
"\C-x\"": "\"\"\C-b"
# inserta una barra invertida (probando escapes de
# barras invertidas en secuencias y macros)
"\C-x\\": "\\"
# Entrecomilla la palabra actual o previa
"\C-xq": "\eb\"\ef\""
# Añade una asociación para refrescar la línea, la cual no está asociada
"\C-xr": redraw-current-line
# Edita variable en línea actual.
"\M-\C-v": "\C-a\C-k$\C-y\M-\C-e\C-a\C-y="
$endif

# usa un timbre visible si hay uno disponible
set bell-style visible

# no cortes carácteres a 7 bits al leer
set input-meta on

# permite que sean insertados caracteres iso-latin1
# en vez de convertidos a secuencias de meta prefijadas
set convert-meta off

# muestra caracteres con el octavo bit establecido
# directamente en vez de como caracteres
# prefijados por meta
set output-meta on

# si hay más de 150 posibles compleciones para una
# palabra, pregunta al usuario si quiere verlas todas
```

```
set completion-query-items 150

# Para FTP
$if Ftp
"\C-xg": "get \M-?"
"\C-xt": "put \M-?"
"\M-.": yank-last-arg
$endif
```

8.4 Órdenes Asociables de Readline

Esta sección describe las órdenes de Readline que pueden estar asociadas a secuencias de caracteres. Puede listar sus asociaciones de caracteres ejecutando `bind -P` o, para un formato más terso, válido para un archivo *inputrc*, `bind -p`. (Véase Sección 4.2 [Instrucciones Integradas de Bash], página 56.) Por defecto, los nombres de órdenes sin una secuencia de teclas que los acompañen no están asociados.

En las siguientes descripciones, *punto* se refiere a la posición actual del cursor, y *marca* se refiere a una posición del cursor guardada por la orden `set-mark`. Se hace referencia al texto entre el punto y la marca como *región*.

8.4.1 Órdenes para Moverse

`beginning-of-line (C-a)`
: Se mueve al inicio de la línea actual.

`end-of-line (C-e)`
: Se mueve al final de la línea.

`forward-char (C-f)`
: Se mueve adelante un carácter.

`backward-char (C-b)`
: Se mueve atrás un carácter.

`forward-word (M-f)`
: Se mueve adelante hasta el final de la siguiente palabra. Las palabras se componen de letras y dígitos.

`backward-word (M-b)`
: Se mueve atrás hasta el principio de la siguiente o la anterior palabra. Las palabras están compuestas de letras y dígitos.

`shell-forward-word ()`
: Se mueve adelante hasta el final de la siguiente palabra. Las palabras están delimitadas por metacaracteres del intérprete sin entrecomillar.

`shell-backward-word ()`
: Se mueve atrás al comienzo de la actual o siguiente palabra. Las palabras están delimitadas por metacaracteres del intérprete sin entrecomillar.

`previous-screen-line ()`
: Trata de mover el punto a la misma columna de pantalla en la anterior línea física de pantalla. Esto no tendrá el efecto deseado si la línea actual de Readline

no ocupa más de una línea física o si el punto no es mayor que la longitud del prompt más la anchura de la pantalla.

`next-screen-line ()`
Trata de mover el punto a la misma columna de pantalla en la siguiente línea física de pantalla. Esto no tendrá el efecto deseado si la línea actual de Readline no ocupa más de una línea física o si la longitud de la actual línea de Readline no es mayor que la longitud del prompt más la anchura de la pantalla.

`clear-screen (C-l)`
Limpia la pantalla y vuelve a dibujar la línea actual, dejando la línea actual en la parte superior de la pantalla.

`redraw-current-line ()`
Recarga la línea actual. Por defecto, esto está sin asociar.

8.4.2 Órdenes para Manipular el Historial

`accept-line (Nueva Línea o Retorno)`
Acepta la línea sin importar dónde esté el cursor. Si esta línea no está vacía, la añade a la lista del historial según los ajustes de las variables `HISTCONTROL` y `HISTIGNORE`. Si esta línea es una línea del historial modificada, entonces restablece la línea del historial a su estado original.

`previous-history (C-p)`
Se mueve 'atrás' a través de la lista del historial, obteniendo anteriores instrucciones.

`next-history (C-n)`
Se mueve 'adelante' a través de la lista del historial, obteniendo la siguiente instrucción.

`beginning-of-history (M-<)`
Se mueve a la primera línea en el historial.

`end-of-history (M->)`
Se mueve al final de la entrada del historial, es decir, la línea que está siendo introducida actualmente.

`reverse-search-history (C-r)`
Busca hacia tras empezando en la línea actual y moviéndose 'arriba' a través del historial según sea necesario. Esto es una búsqueda incremental.

`forward-search-history (C-s)`
Busca hacia delante empezando en la línea actual y moviéndose 'abajo' a través del historial según sea necesario. Esto es una búsqueda incremental.

`non-incremental-reverse-search-history (M-p)`
Busca hacia tras empezando en la línea actual y moviéndose 'arriba' a través del historial según sea necesario usando una búsqueda no incremental para una cadena proporcionada por el usuario. La cadena de búsqueda puede coincidir en cualquier lugar en una línea del historial.

`non-incremental-forward-search-history (M-n)`
: Busca hacia delante empezando en la línea actual y moviéndose 'abajo' a través del historial según sea necesario usando una búsqueda no incremental para una cadena proporcionada por el usuario. La cadena de búsqueda puede coincidir en cualquier lugar en una línea del historial.

`history-search-forward ()`
: Busca hacia delante a través del historial para la cadena de caracteres entre el comienzo de la línea actual y el punto. La cadena de búsqueda debe coincidir al principio de la línea del historial. Esto es una búsqueda no incremental. Por defecto, esta instrucción no está asociada.

`history-search-backward ()`
: Busca hacia tras a través del historial para la cadena de caracteres entre el comienzo de la línea actual y el punto. La cadena de búsqueda debe coincidir al principio de la línea del historial. Esto es una búsqueda no incremental. Por defecto, esta instrucción no está asociada.

`history-substring-search-forward ()`
: Busca hacia delante a través del historial para la cadena de caracteres entre el comienzo de la línea actual y el punto. La cadena de búsqueda puede coincidir en cualquier lugar de la línea del historial. Esto es una búsqueda no incremental. Por defecto, esta instrucción no está asociada.

`history-substring-search-backward ()`
: Busca hacia tras a través del historial para la cadena de caracteres entre el comienzo de la línea actual y el punto. La cadena de búsqueda puede coincidir en cualquier lugar de la línea del historial. Esto es una búsqueda no incremental. Por defecto, esta instrucción no está asociada.

`yank-nth-arg (M-C-y)`
: Inserta el primer argumento para la instrucción previa (normalmente la segunda palabra en la línea anterior) en el punto. Con un argumento *n*, inserta la palabra número *n* de la instrucción anterior (las palabras en la instrucción anterior comienzan con palabra 0). Un argumento negativo inserta la palabra número *n* desde el final de la instrucción previa. Una vez que sea computado el argumento *n*, se extrae el argumento como si se hubiera especificado la expansión del historial '`!n`'.

`yank-last-arg (M-. or M-_)`
: Inserta el último argumento de la instrucción anterior (la última palabra de la anterior entrada del historial). Con un argumento numérico, se comporta exactamente como `yank-nth-arg`. Las llamadas sucesivas a `yank-last-arg` se mueven atrás a través de la lista del historial, insertando la última palabra (o la palabra especificada por el argumento de la primera llamada) de cada línea en orden. Cualquier argumento numérico proporcionado a estas llamadas sucesivas determina la dirección a la que moverse por el historial. Un argumento negativo alterna la dirección por el historial (atrás o adelante). Se usan las herramientas de expansión del historial para extraer el último argumento, como si se hubiera especificado la expansión del historial '`!$`'.

8.4.3 Órdenes para Cambiar Texto

end-of-file (usually C-d)
: El carácter que indica fin-de-fichero como establece, por ejemplo, `stty`. Si se lee este carácter cuando no hay caracteres en la línea y el punto está al principio de la línea, Readline lo interpreta como el final de entrada y devuelve EOF.

delete-char (C-d)
: Borra el carácter en el punto. Si está función está asociada al mismo carácter que el carácter tty EOF, como está comúnmente *C-d*, consulte arriba los efectos.

backward-delete-char (Rubout)
: Borra el carácter antes del cursor. Un argumento numérico significa cortar los caracteres en vez de borrarlos.

forward-backward-delete-char ()
: Borra el carácter bajo el cursor, a no ser que el cursor esté al final de la línea, en cuyo caso es borrado el carácter antes del cursor. Por defecto, esto no está asociado a una tecla.

quoted-insert (C-q or C-v)
: Añade el siguiente carácter tecleado a la línea literal. Esto es cómo insertar secuencias de teclas como *C-q*, por ejemplo.

self-insert (a, b, A, 1, !, ...)
: Inserte por sí mismo.

bracketed-paste-begin ()
: Esta función está pensada para estar asociada a la secuencia de escape bracketed paste enviada por algunas terminales, y tal asociación está asignada por defecto. Permite a Readline insertar el texto pegado como una única unidad sin tratar cada carácter como si hubiera sido leído del teclado. Los caracteres se insertan como si cada uno estuviera asociado a `self-insert` en vez de ejecutar las órdenes de edición.

transpose-chars (C-t)
: Arrastra el carácter antes del cursor adelante del carácter en el cursor, moviendo también el cursor adelante. Si el punto de inserción está al final de la línea, esto transpone los últimos dos caracteres de la línea. Los argumentos negativos no tienen efecto.

transpose-words (M-t)
: Arrastra la palabra antes del punto tras la palabra después del punto, moviendo el punto también tras esa palabra. Si el punto de inserción está al final de la línea, esto traspone las dos últimas palabras en la línea.

upcase-word (M-u)
: Pone en mayúscula la actual (o siguiente) palabra. Con un argumento negativo, pone en mayúscula la palabra anterior, pero no mueve el cursor.

downcase-word (M-l)
: Pone en minúscula la actual (o siguiente) palabra. Con un argumento negativo, pone en minúscula la palabra anterior, pero no mueve el cursor.

`capitalize-word (M-c)`
: Pone la primera letra de la actual (o siguiente) palabra en mayúscula. Con un argumento negativo se aplica a la palabra anterior, pero no mueve el cursor.

`overwrite-mode ()`
: Alterna el modo de sobrescritura. Con un argumento numérico positivo explícito, cambia al modo de sobrescritura. Con un argumento no positivo explícito, cambia al modo de inserción. Esta orden solo afecta al modo `emacs`; el modo `vi` sobrescribe de forma diferente. Cada llamada a `readline()` empieza en modo de inserción.

  En el modo de sobrescritura, los caracteres asociados a `self-insert` reemplazan el texto en el punto en vez de empujar el texto a la derecho. Los caracteres asociados a `backward-delete-char` reemplazan el carácter antes del punto con un espacio.

  Por defecto, esta orden no está asociada.

8.4.4 Cortar y Pegar

`kill-line (C-k)`
: Corta el texto desde punto hasta el final de la línea.

`backward-kill-line (C-x Rubout)`
: Corta atrás desde el cursor hasta el principio de la línea actual.

`unix-line-discard (C-u)`
: Corta atrás desde el cursor hasta el principio de la línea actual.

`kill-whole-line ()`
: Corta todos los caracteres en la línea actual, no importa dónde esté el punto. Por defecto, esto está sin asociar.

`kill-word (M-d)`
: Corta desde el punto hasta el final de la palabra actual, o si está entre palabras, hasta el final de la siguiente palabra. Los límites de palabra son los mismos que con `forward-word`.

`backward-kill-word (M-SUPR)`
: Corta la palabra antes del punto. Los límites de palabra son los mismos que con `backward-word`.

`shell-kill-word ()`
: Corta desde el punto hasta el final de la palabra actual o, cuando entre palabras, hasta el final de la siguiente palabra. Los límites de palabra son los mismos que con `shell-forward-word`.

`shell-backward-kill-word ()`
: Corta la palabra antes del punto. Los límites de palabra son los mismos que los de `shell-backward-word`.

`unix-word-rubout (C-w)`
: Corta la palabra antes del punto, usando el espacio en blanco como un límite de palabra. El texto cortado se guarda en el anillo de corte.

`unix-filename-rubout ()`
: Corta la palabra antes del punto, usando el espacio en blanco y el carácter de barra como los límites de palabra. El texto cortado se guarda en el anillo de corte.

`delete-horizontal-space ()`
: Elimina todos los espacios y tabulaciones alrededor del punto. Por defecto, esto está sin asociar.

`kill-region ()`
: Corta el texto en la región actual. Por defecto, esta orden está sin asociar.

`copy-region-as-kill ()`
: Copia el texto de la región al búfer de corte, para que pueda ser pegado inmediatamente. Por defecto, esta orden está sin asignar.

`copy-backward-word ()`
: Copia la palabra antes del punto al búfer de corte. Los límites de palabra son los mismos que los de `backward-word`. Por defecto, esta orden no está asociada.

`copy-forward-word ()`
: Copia la palabra que sigue al punto al búfer de corte. Los límites de palabra son los mismos que los de `backward-word`. Por defecto, esta orden no está asociada.

`yank (C-y)`
: Pega lo superior del anillo de corte en el búfer en el punto.

`yank-pop (M-y)`
: Rota el kill-ring y pega lo que hay nuevo arriba. Solo puede hacer esto si la anterior orden es `yank` o `yank-pop`.

8.4.5 Especificando Argumentos Numéricos

`digit-argument (`*`M-0`*`, `*`M-1`*`, ... `*`M--`*`)`
: Añade este dígito al argumento que ya se esté acumulando o empieza un nuevo argumento. *M--* empieza un argumento negativo.

`universal-argument ()`
: Esto es otro modo de especificar un argumento. Si esta orden es seguida de uno o más dígitos, opcionalmente con un signo de menos inicial, esos dígitos definen el argumento. Si la orden es seguida por dígitos, ejecutar `universal-argument` de nuevo acaba el argumento numérico, pero es ignorado de lo contrario. Como caso especial, si esta orden es inmediatamente seguida por un carácter que no es un dígito ni un signo de menos, la cuenta de argumentos para la siguiente instrucción se multiplica por cuatro. La cuenta de argumentos es inicialmente uno, así que ejecutar esta función la primera vez hace la cuenta de argumentos cuatro, una segunda vez hace la cuenta de argumentos dieciséis y así sucesivamente. Por defecto, no está asociada a una tecla.

8.4.6 Dejar a Readline Escribir por Usted

`complete (TAB)`
: Trata de realizar una compleción en el texto antes del punto. La compleción real realizada es especifica de la aplicación. Bash intenta la compleción tratando

el texto como una variable (si el texto empieza por '$'), nombre de usuario (si el texto empieza por '~'), nombre de anfitrión (si el texto empieza por '@') o instrucción (incluyendo aliases y funciones) en orden. Si ninguna de estas produce una coincidencia, se intenta la compleción de nombre de archivo.

`possible-completions (M-?)`
: Lista las posibles compleciones del texto antes del punto. Al mostrar compleciones, Readline establece el número de columnas usadas para mostrar el valor de `completion-display-width`, el valor de la variable de entorno `COLUMNS` o la anchura de la pantalla, en ese orden.

`insert-completions (M-*)`
: Inserta todas las compleciones de texto antes del punto que habrían sido generadas por `possible-completions`.

`menu-complete ()`
: Similar a `complete`, pero reemplaza la palabra que se va a completar con una única coincidencia de la lista de posibles compleciones. La ejecución repetida de `menu-complete` recorre la lista de posibles compleciones, insertando cada coincidencia en orden. Al final de la lista de compleciones, suena el timbre (sujeto al ajuste de `bell-style`) y se restablece el texto original. Un argumento de *n* se mueve *n* posiciones adelante en la lista de coincidencias; un argumento negativo se puede usar para moverse atrás a través de la lista. Esta instrucción está pensada para ser asociada a `TAB`, pero por defecto no está asociada.

`menu-complete-backward ()`
: Idéntica a `menu-complete`, pero se mueve atrás a través de la lista de posibles compleciones, como si a `menu-complete` se le hubiera dado un argumento negativo.

`delete-char-or-list ()`
: Elimina el carácter bajo el cursor si no está al principio o al final de la línea (como `delete-char`). Si está al final de la línea, se comporta de forma idéntica a `possible-completions`. Esta orden está sin asignar por defecto.

`complete-filename (M-/)`
: Intenta la compleción de nombre de archivo en el texto antes del punto.

`possible-filename-completions (C-x /)`
: Lista las posibles compleciones del texto antes del punto, tratándolo como un nombre de archivo.

`complete-username (M-~)`
: Intenta la compleción en el texto antes del punto, tratándolo como un nombre de archivo.

`possible-username-completions (C-x ~)`
: Lista las posibles compleciones del texto antes del punto, tratándolo como un nombre de usuario.

`complete-variable (M-$)`
: Intenta la compleción en el texto antes del punto, tratándolo como una variable del intérprete.

`possible-variable-completions (C-x $)`
: Lista las posibles compleciones del texto antes del punto, tratándolo como una variable del intérprete.

`complete-hostname (M-@)`
: Intenta la compleción del texto antes del punto, tratándolo como un nombre de anfitrión.

`possible-hostname-completions (C-x @)`
: Lista las posibles compleciones del texto antes del punto, tratándolo como un nombre de anfitrión.

`complete-command (M-!)`
: Intenta la compleción en el texto antes del punto, tratándolo como un nombre de instrucción. La compleción de instrucciones trata de hacer coincidir el texto con aliases, palabras reservadas, funciones del intérprete, instrucciones integradas del intérprete y finalmente nombres de archivo ejecutables, en ese orden.

`possible-command-completions (C-x !)`
: Lista las posibles compleciones del texto antes del punto, tratándolo como un nombre de orden.

`dynamic-complete-history (M-TAB)`
: Intenta la compleción del texto antes del punto, comparándo el texto con líneas de la lista del historial en busca de posibles coincidencias de compleción.

`dabbrev-expand ()`
: Intenta la compleción de menú en el texto antes del punto, comparando el texto con las líneas de la lista del historial en busca de posibles coincidencias de compleción.

`complete-into-braces (M-{)`
: Realiza la compleción de nombre de archivo e inserta la lista de compleciones posibles entre llaves para que la lista esté disponible para el intérprete (véase Sección 3.5.1 [Expansión de Llaves], página 25).

8.4.7 Macros de Teclado

`start-kbd-macro (C-x ()`
: Empieza a guardar los caracteres escritos en la macro actual de teclado.

`end-kbd-macro (C-x ))`
: Deja de guardar los caracteres escritos en la macro actual de teclado y guarda la definición.

`call-last-kbd-macro (C-x e)`
: Vuelve a ejecutar la última macro de teclado definida, haciendo que los caracteres en la macro aparezcan como si fueran teclados en el teclado.

`print-last-kbd-macro ()`
: Imprime la última macro de teclado definida en un forma compatible con el archivo *inputrc*.

8.4.8 Algunas Órdenes Variadas

`re-read-init-file (C-x C-r)`
: Lee en el contenido del archivo *inputrc* e incorpora las asociaciones o asignaciones de variable encontradas ahí.

`abort (C-g)`
: Aborta la orden que se está editando actualmente y suena el timbre de la terminal (sujeto al ajuste de `bell-style`).

`do-lowercase-version (M-A, M-B, M-x, ...)`
: Si el carácter metaficado *x* está en mayúscula, ejecuta la instrucción que está asociada al carácter metaficado correspondiente en minúscula. El comportamiento no está definido si *x* ya está en minúscula.

`prefix-meta (ESC)`
: Metafica el siguiente carácter escrito. Esto es para teclados sin una tecla meta. Escribir '`ESC f`' es equivalente a teclear *`M-f`*.

`undo (C-_ or C-x C-u)`
: Deshacer incremental, recordado separadamente para cada línea.

`revert-line (M-r)`
: Deshace todos los cambios hechos a esta línea. Esto es como ejecutar la orden `undo` las veces suficientes para volver al comienzo.

`tilde-expand (M-&)`
: Realiza la expansión de virgulilla en la palabra actual.

`set-mark (C-@)`
: Establece la marca en el punto. Si se proporciona un argumento numérico, la marca se establece en esa posición.

`exchange-point-and-mark (C-x C-x)`
: Intercambia el punto con la marca. La posición del cursor actual se establece a la posición guardada, y la posición antigua del cursor se guarda como la marca.

`character-search (C-])`
: Un carácter es leído, y el punto se mueve a la siguiente ocurrencia de ese carácter. Una cuenta negativa busca ocurrencias previas.

`character-search-backward (M-C-])`
: Un carácter es leído, y el punto se mueve a la anterior ocurrencia de ese carácter. Una cuenta negativa busca ocurrencias posteriores.

`skip-csi-sequence ()`
: Lee suficientes caracteres para consumir una secuencia multitecla como aquellas definidas por teclas como Inicio y Fin. Estas secuencias comienzan con un Indicador de Secuencia de Control (ISC), normalmente ESC-[. Si esta secuencia está asociada a `"\e["`, las teclas que produzcan tales secuencias no tendrán efecto a no ser que sean específicamente asociadas a una orden readline, en vez de insertar caracteres inconexos en el búfer de edición. Esto está sin asociar por defecto, pero normalmente asociado a ESC-[.

`insert-comment (M-#)`
: Sin un argumento numérico, el valor de la variable `comment-begin` se inserta al comienzo de la línea actual. Si se proporciona un argumento numérico, esta orden actúa como un centinela: si los caracteres al comienzo de la línea no coinciden con el valor de `comment-begin`, el valor es insertado; de lo contrario, los caracteres en `comment-begin` son eliminados del comienzo de la línea. En ambos casos, la línea se acepta como si se hubiera tecleado una nueva línea. El valor predeterminado de `comment-begin` hace que esta orden haga la línea actual un comentario del intérprete. Si un argumento numérico hace que el carácter de comentario sea eliminado, el línea será ejecutada por el intérprete.

`dump-functions ()`
: Imprime todas las funciones y sus asociaciones de teclas en el flujo de salida de Readline. Si se proporciona un argumento numérico, la salida se formatea en una forma que se puede hacer parte de un archivo *inputrc*. Esta orden está sin asociar por defecto.

`dump-variables ()`
: Imprime todas las variables que pueden ser establecidas y sus valores al flujo de salida de Readline. Si se proporciona un argumento numérico, la salida se formatea de una forma que se puede hacer parte de un archivo *inputrc*. Esta orden está sin asociar por defecto.

`dump-macros ()`
: Imprime todas las secuencias de teclas de Readline asociadas a macros y las cadenas que producen. Si se proporciona un argumento numérico, la salida se formatea de una forma que se puede hacer parte de un archivo *inputrc*. Esta orden está sin asociar por defecto.

`glob-complete-word (M-g)`
: La palabra antes del punto se trata como un patrón para la expansión de nombre de patrón, con un asterisco añadido implícitamente. Este patrón se usa para generar una lista de nombres de archivos coincidentes para posibles compleciones.

`glob-expand-word (C-x *)`
: La palabra antes del punto se trata como un patrón para la compleción de nombre de archivo, y es insertada la lista de nombres de archivos coincidentes, reemplazando la palabra. Si se proporciona un argumento numérico, se añade un '`*`' antes de la expansión de nombre de archivo.

`glob-list-expansions (C-x g)`
: Se muestra la lista de expansiones que han sido generadas por `glob-expand-word`, y se vuelve a dibujar la línea. Si se proporciona un argumento numérico, se añade un '`*`' antes de la expansión de nombre de ruta.

`display-shell-version (C-x C-v)`
: Muestra la información de versión de la instancia actual de Bash.

`shell-expand-line (M-C-e)`
: Expande la línea como hace el intérprete. Esto realiza la expansión de alias y de historial así como todas las expansiones de palabra del intérprete (véase Sección 3.5 [Expansiones del Intérprete], página 24).

`history-expand-line (M-^)`
: Realiza la expansión de historial en la línea actual.

`magic-space ()`
: Realiza la expansión de historial en la línea actual e inserta un espacio (véase Sección 9.3 [Interacción con el Historial], página 163).

`alias-expand-line ()`
: Realiza la expansión de alias en la línea actual (véase Sección 6.6 [Aliases], página 105).

`history-and-alias-expand-line ()`
: Realiza expansión de alias y de historial en la línea actual.

`insert-last-argument (M-. or M-_)`
: Un sinónimo de `yank-last-arg`.

`operate-and-get-next (C-o)`
: Acepta la línea actual para ejecución y obtiene la siguiente línea relativa a la línea actual del historial para editar. Un argumento numérico, si se proporciona, especifica la entrada del historial que usar en vez de la línea actual.

`edit-and-execute-command (C-xC-e)`
: Llama a un editor en la línea actual de orden y ejecuta el resultado como instrucciones del intérprete. Bash trata de llamar a `$VISUAL`, `$EDITOR` y `emacs` como el editor, en ese orden.

8.5 Modo vi de Readline

Si bien la biblioteca Readline tiene un conjunto completo de las funciones de edición de `vi`, contiene lo suficiente para permitir la edición simple de la línea. El modo `vi` de Readline se comporta como se especifica en el estándar POSIX.

Para alternar interactivamente entre los modos de edición de `emacs` y `vi`, use las instrucciones '`set -o emacs`' y '`set -o vi`' (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68). El predeterminado de Readline es el modo `emacs`.

Cuando introduces una línea en el modo `vi`, ya está metido en el modo de 'inserción', como si hubiera escrito '`i`'. Pulsar `ESC` le cambia al modo de 'órdenes', donde puede editar el texto de la línea con las teclas de movimiento estándares de `vi`, moverse a líneas del historial previas con '`k`' y a líneas siguientes con '`j`' y demás.

8.6 Compleción Programable

Cuando se intenta la compleción de palabra para un argumento de una instrucción para la que ha sido definida una especificación de compleción (una *compspec*) usando la instrucción integrada `complete` (véase Sección 8.7 [Instrucciones Integradas de Compleción Programable], página 153), se llaman a las utilidades de compleción.

Primero se identifica el nombre de la instrucción. Si se ha definido una compspec para esa instrucción, la compspec se usa para generar la lista de posibles compleciones para la palabra. Si la palabra de instrucción es la cadena vacía (compleción intentada al principio de una línea vacía), se usa cualquier compspec definida con la opción `-E` para `complete`. Si la palabra de instrucción es un nombre de ruta completo, se busca primero una compspec para el nombre de ruta completo. Si no se encuentra una compspec para el nombre de ruta completo, se hace un intento para encontrar una comspec para la porción que sigue a la barra final. Si ninguna de esas búsquedas produce una compspec, cualquier compspec definida con la opción `-D` de `complete` se usa como la predeterminada. Si no hay compspec predeterminada, Bash trata la expansión de alias en la palabra de instrucción como un último recurso y trata de encontrar una compspec para la palabra de instrucción desde cualquier expansión exitosa

Una vez que se haya encontrado una compspec, se usa para generar la lista de palabras coincidentes. Si no se ha encontrado una compspec, se realiza la compleción predeterminada de Bash descrita anteriormente (véase Sección 8.4.6 [Órdenes para Compleción], página 145).

Primero se usan las acciones especificadas por la compspec. Solo son devueltas las coincidencias prefijadas por la palabra que está siendo completada. Cuando se usa la opción `-f` o `-d` para la compleción de nombre de archivo o directorio, se usa la variable del intérprete `FIGNORE` para filtrar las coincidencias. Véase Sección 5.2 [Variables de Bash], página 82, para una descripción de `FIGNORE`.

Todas las compleciones especificadas por un patrón de expansión de nombre de archivo para la opción `-G` son generadas después. Las palabras generadas por el patrón no necesitan coincidir con la palabra que está siendo completada. La variable del intérprete `GLOBIGNORE` no se usa para filtrar las coincidencias, sino que se usa la variable del intérprete `FIGIGNORE`.

A continuación, se considera la cadena especificada como el argumento para la opción `-W` La cadena se divide primero usando los caracteres en la variable especial `IFS` como delimitadores. Se respeta el entrecomillado del intérprete dentro de la cadena, para proporcionar un mecanismo para que las palabras contengan metacaracteres del intérprete o caracteres en el valor de `IFS`. Cada palabra es entonces expandida usando la expansión de llaves, expansión de virgulilla, expansión de parámetro y variable, sustitución de instrucción y expansión aritmética, como se describe anteriormente (véase Sección 3.5 [Expansiones del Intérprete], página 24). Los resultados se dividen usando las reglas descritas anteriormente (véase Sección 3.5.7 [División de Palabras], página 34). Los resultados de la expansión son coincididos con los prefijos con la palabra que está siendo completada, y las palabras coincidentes se convierten en las posibles compleciones.

Después de que estas coincidencias hayan sido generadas, se llama a cualquier función o instrucción del intérprete especificada con las opciones `-F` y `-C`. Cuando se llama a la instrucción o función, se asignan valores a las variables `COMP_LINE`, `COMP_POINT`, `COMP_KEY` y `COMP_TYPE` como se describe anteriormente (véase Sección 5.2 [Variables de Bash], página 82). Si se está llamando a una función del intérprete, se establecen también las variables `COMP_WORDS` y `COMP_CWORD`. Cuando se llama a la función o instrucción, el primer argumento ($1) es el nombre de la instrucción cuyos argumentos están siendo completados, el segundo argumento ($2) es la palabra que está siendo completada y el tercer argumento ($3) es la palabra que precede a la palabra que está siendo completada en la línea de órdenes actual. No se realiza ningún filtrado de las compleciones generadas sobre la palabra

que está siendo completada; la función o instrucción tiene libertad absoluta para generar las coincidencias.

Se llama primero a cualquier función especificada con `-F`. La función puede usar cualquiera de las herramientas del intérprete, incluyendo las instrucciones integradas `compgen` y `compopt` descritas a continuación (véase Sección 8.7 [Instrucciones Integradas de Compleción Programable], página 153), para generar las coincidencias. Debe poner las posibles compleciones en la variable de vector `COMPREPLY`, una por elemento de vector.

A continuación, se llama a cualquier instrucción especificada con la opción `-C` en un entorno equivalente a la sustitución de instrucción. Debería imprimir una lista de compleciones, una por línea, en la salida estándar. La barra invertida puede ser usada para escapar una nueva línea, si es necesario.

Después de que sean generadas todas las posibles compleciones, se aplica a la lista cualquier filtro especificado con la opción `-X`. El filtro es un patrón como el que se usa para la expansión de nombre de archivo; un '`&`' en el patrón es reemplazado por el texto de la palabra que está siendo completada. Un '`&`' literal se puede escapar con una barra invertida; se elimina la barra invertida antes de intentar una coincidencia. Cualquier compleción que coincida con el patrón será eliminada de la lista. Un '`!`' inicial niega el patrón; en este caso se eliminará cualquier compleción que no coincida con el patrón. Si la opción del intérprete `nocasematch` (vea la descripción de `shopt` en Sección 4.3.2 [La Instrucción Integrada Shopt], página 73) está activada, la coincidencia se realiza sin tener en cuenta las mayúsculas y minúsculas de caracteres alfabéticos.

Finalmente, los prefijos y sufijos especificados con las opciones `-P` y `-S` se añaden a cada miembro de la lista de compleción, y el resultado es devuelto al código de compleción de Readline como la lista de compleciones posibles.

Si las acciones aplicadas anteriormente no generan ningún resultado y la opción `-o dirnames` fue proporcionada a `complete` cuando compspec fue definida, se intenta la compleción de nombre de directorio.

Si la opción `-o plusdirs` fue proporcionada a `complete` cuando la compspec fue definida, se intenta la compleción de nombre de directorio y las coincidencias se añaden a los resultados de las otras acciones.

Por defecto, si se encuentra una compspec, lo que sea que genere es devuelto al código de compleción como el conjunto completo de compleciones posibles. Las compleciones predeterminadas de Bash no se intentan, y se deshabilita el valor predeterminado de Readline de compleción de nombre de archivo. Si la opción `-o bashdefault` fue proporcionada a `complete` cuando la compspec fue definida, se intentan las compleciones predeterminadas de Bash si compspec no genera coincidencias. Si la opción `-o default` fue proporcionada a `complete` cuando la compspec fue definida, la compleción predeterminada de Readline se realizará si la compspec (y, si se intenta, las compleciones predeterminadas de Bash) no genera coincidencias.

Cuando una compspec indica que se desea la compleción de nombre de directorio, las funciones de compleción programables fuerza a Readline a añadir una barra a los nombres completados que son enlaces simbólicos a directorios, sujetos al valor de la variable de Readline *mark-directories*, sin importar el ajuste de la variable de Readline *mark-symlinked-directories*.

Hay algún soporte para modificar dinámicamente compleciones. Esto es muy útil cuando se usa en conjunto con una compleción predeterminada especificada con `-D`. Es posible para las funciones del intérprete ser ejecutadas como manejadores de compleción para indicar que la compleción debería ser reintentada devolviendo un estado de salida de 124. Si una función del intérprete devuelve 124 y cambia la compspec asociada con la instrucción en que la compleción se está intentando (proporcionada como el primer argumento cuando se ejecuta la función), la compleción programable se reinicia desde el principio, con un intento de encontrar una nueva compspec para esa instrucción. Esto permite que un conjunto de compleciones sea construido dinámicamente mientras se intenta la compleción, en vez de ser cargado de golpe.

Por ejemplo, asumiendo que hubiera una biblioteca de compspecs, cada una en un archivo correspondiente al nombre de la instrucción, la siguiente compleción predeterminada cargaría compleciones dinámicamente:

```
_completion_loader()
{
    . "/etc/bash_completion.d/$1.sh" >/dev/null 2>&1 && return 124
}
complete -D -F _completion_loader -o bashdefault -o default
```

8.7 Instrucciones Integradas de Compleción Programable

Hay disponibles tres instrucciones integradas para manipular las herramientas de compleción programables: una para especificar cómo van a ser completados los argumentos para una instrucción particular, y dos para modificar la compleción mientras ocurre.

`compgen`

`compgen [`*opción*`] [`*palabra*`]`

Genera posibles coincidencias de compleción para *palabra* de acuedo a las *opciones*, que pueden ser cualquier opción aceptada por la instrucción integrada `complete` con la excepción de `-p` y `-r`, y escribe las coincidencias a la salida estándar. Al usar las opciones `-F` o `-C`, las distintas variables del intérprete establecidas por las herramientas de compleción programables, mientras estén disponibles, no tendrán valores útiles.

Las coincidencias serán generadas de la misma forma que si el código de la compleción programable las hubiera generado directamente de una especificación de compleción con las mismas banderas. Si se especifica *palabra*, solo se mostrarán aquellas compleciones que coincidan con *palabra*.

El valor de retorno es verdadero a no ser que se proporcione una opción inválida o no se hayan generado coincidencias.

`complete`

`complete [-abcdefgjksuv] [-o` *opción-comp*`] [-DEI] [-A` *acción*`] [-G` *rutglob*`] [-W` *lista-palabras*`][-F` *función*`] [-C` *instrucción*`] [-X` *filtrapat*`]`
`[-P` *prefijo*`] [-S` *sufijo*`]` *nombre* `[`*nombre ...*`]`
`complete -pr [-DEI] [`*nombre ...*`]`

Especifica cómo deberían ser completados los argumentos para cada *nombre*. Si se proporciona la opción `-p`, o si no se proporcionan opciones, se imprimen las especificaciones de compleción existentes de un modo que permite que puedan ser reutilizados como entrada. La opción `-r` elimina una especificación de compleción por cada *nombre* o, si no se proporcionan *nombre*s, todas las especificaciones de compleción. La opción `-D` indica que otras opciones y acciones proporcionadas se deberían aplicar a la instrucción de compleción "default"; esto es, la compleción intentada en una instrucción para la que no ha sido definida una compleción previamente. La opción `-E` indica que otras opciones y acciones proporcionadas deberían aplicarse a la instrucción de compleción "empty"; esto es, la compleción intentada en una línea en blanco. La opción `-l` indica que otras acciones y opciones proporcionadas deberían aplicarse a la compleción en la palabra inicial de no asignación en la línea o después de un delimitador de instrucción como '`;`' o '`|`', que es normalmente el nombre de compleción de instrucción. Si se proporcionan varias opciones, la opción `-D` tiene precedencia sobre `-E`, y ambas tienen preferencia sobre `-l`. Si se proporciona cualquiera de las opciones `-D`, `-E` o `-l`, se ignoran cualquier otras argumentos *nombre*; estas compleciones solo se aplican al caso especificado por la opción.

El proceso de aplicar estas especificaciones de compleción cuando se intenta la compleción de palabra se describe más arriba (véase Sección 8.6 [Compleción Programable], página 150).

Otras opciones, si se especifican, tienen los siguientes significados. Los argumentos para las opciones `-G`, `-W`, and `-X` (y, si es necesario, las opciones `-P` y `-S`) deben ser entrecomillados para protegerlos de la expansión antes de que sea llamada la instrucción integrada `complete`.

`-o` *opción-comp*

La *opción-comp* controla varios aspectos del comportamiento de compspec más allá de la simple generación de compleciones. *opción-comp* puede ser una de:

`bashdefault`

Realiza el resto de las compleciones predeterminadas de Bash si la compspec no genera coincidencias.

`default` Una la compleción de nombre de archivo predeterminada de Readline si la compspec no genera coincidencias.

`dirnames` Realiza compleción de nombre de directorio si la compspec no genera coincidencias.

`filenames`

Le dice a Readline que la compspec genera nombres de archivo, para que pueda realizar cualquier procesamiento específico para nombres de archivo (como añadir una barra a nombres de directorio, entrecomillar caracteres especiales o suprimir los espacios finales). Esta opción está pensada para usarse con la función del intérprete especificada con `-F`.

`noquote` Le dice a Readline que no entrecomille las palabras completadas si son nombres de archivo (entrecomillar nombres de archivo es lo predeterminado).

`nosort` Le dice a Readline que no ordene la lista de posibles compleciones alfabéticamente.

`nospace` Le dice a Readline que no añada un espacio (lo predeterminado) a palabras completadas al final de la línea.

`plusdirs` Después de que las coincidencias definidas por la compspec hayan sido generadas, se intenta la compleción de nombre de directorio y se añaden las coincidencias a los resultados de las otras acciones.

`-A` *acción* La *acción* puede ser una de las siguientes para generar una lista de posibles compleciones:

`alias` Nombres de alias. También puede ser especificada como `-a`.

`arrayvar` Nombres de variable de vector.

`binding` Nombres de asociación de teclas de Readline (véase Sección 8.4 [Órdenes Asociables de Readline], página 140).

`builtin` Nombres de instrucciones integradas del intérprete. También puede ser especificado como `-b`.

`command` Nombres de instrucciones. También puede ser especificado como `-c`.

`directory` Nombres de directorio. También puede ser especificado como `-d`.

`disabled` Nombres de instrucciones integradas deshabilitadas.

`enabled` Nombres de instrucciones integradas habilitadas.

`export` Nombres de variables del intérprete exportadas. También puede ser especificado como `-e`.

`file` Nombres de archivo. También puede ser especificado como `-f`.

`function` Nombres de funciones del intérprete.

`group` Nombres de grupo. También puede ser especificado como `-g`.

`helptopic` Temas de ayuda como acepta la instrucción integrada `help` (véase Capítulo 4 [Instrucciones Integradas del Intérprete], página 48).

`hostname` Nombres de anfitrión, como son tomados por el archivo especificado por la variable del intérprete `HOSTNAME` (véase Sección 5.2 [Variables de Bash], página 82).

`job` Nombres de tarea, si el control de tareas está activo. También puede ser especificado como `-j`.

`keyword` Palabras reservadas del intérprete. También puede ser especificado como `-k`.

`running` Nombres de tareas ejecutándose, si el control de tareas está activado.

`service` Nombres de servicios. También puede ser especificado como `-s`.

`setopt` Argumentos válidos para la opción `-o` de la instrucción integrada `set` (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68).

`shopt` Los nombres de opción del intérprete como acepta la instrucción integrada `shopt` (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).

`signal` Nombres de señal.

`stopped` Nombres de tareas detenidas, si está activado el control de tareas.

`user` Nombres de usuario. También puede ser especificado como `-u`.

`variable` Nombres de todas las variables del intérprete. También puede ser especificado como `-v`.

`-C` *instrucción*
instrucción se ejecuta en un entorno de subintérprete y su salida se usa como las posibles compleciones.

`-F` *función*
La función del intérprete *función* se ejecuta en el entorno actual del intérprete. Cuando se ejecuta, $1 es el nombre de la instrucción cuyos argumentos están siendo completados, $2 es la palabra siendo completada, y $3 es la palabra que precede a la palabra siendo completada, como se describe anteriormente (véase Sección 8.6 [Compleción Programable], página 150). Cuando finaliza, se recuperan las posibles compleciones del valor de la variable de vector `COMPREPLY`.

`-G` *patglob*
La expansión de nombre de archivo *patglob* se expande para generar las posibles compleciones.

`-P` *prefijo*
prefijo se añade al comienzo de cada posible compleción después de que se hayan aplicado todas las otras opciones.

`-S` *sufijo* *sufijo* se añade a cada compleción posible después de que se hayan aplicado todas las otras opciones.

`-W` *lista-de-palabras*
La *lista-de-palabras* se divide usando los caracteres en la variable especial `IFS` como delimitadores, y se expande cada palabra resultante. Las posibles compleciones son los miembros de la lista resultante que coinciden con la palabra que está siendo completada.

`-X` *filtrapat*
filtrapat es un patrón como se usa para la expansión de nombre de archivo. Se aplica a la lista de posibles compleciones generada por las opciones y argumentos precedentes, y cada compleción coincidente con *filtrapat* se elimina de la lista. Un '`!`' inicial en *filtrapat* niega el patrón; en este caso, cualquier compleción que no coincide con *filtrapat* es eliminada.

El valor de retorno es verdadero a no ser que se proporcione una opción inválida, una opción diferente de `-p` o `-r` se proporcione sin un argumento *nombre*, se realice un intento para eliminar una especificación de compleción para un *nombre* para el cual no existe ninguna especificación u ocurre un error añadiendo una especificación de compleción.

`compopt`

`opc-comp [-o` *opción*`] [-DEI] [+o` *opción*`] [`*nombre*`]`

Modifica opciones de compleción para cada *nombre* según las *opcione*s, o para la compleción actualmente en ejecución si no se proporcionan *nombre*s. Si no se dan *opcione*s, muestra las opciones de compleción para cada *nombre* o la compleción actual. Los posibles valores de *opción* son aquellos válidos para la instrucción integrada `complete` descrita anteriormente. La opción `-D` indica que otras opciones proporcionadas deberían aplicarse a la compleción de instrucción predeterminada; es decir, la compleción intentada en una instrucción para la cual no se ha definido ninguna compleción anteriormente. La opción `-E` indica que otras opciones proporcionadas deberían aplicarse la compleción de instrucción vacía; es decir, la compleción intentada en una línea en blanco. La opción `-I` indica que otras opciones proporcionadas deberían aplicarse a la compleción en la palabra inicial de no asignación de la línea, o después de un delimitador de instrucción como '`;`' o '`|`', que es normalmente la compleción de nombre de instrucción.

Si se proporcionan varias opciones, la opción `-D` tiene precedencia sobre `-E`, y ambas tienen preferencia sobre `-I`

El valor de retorno es verdadero a no ser que se proporcione una opción inválida, se realice un intento de modificar las opciones para un *nombre* para el que no existe especificación de compleción u ocurra un error de salida.

8.8 Un Ejemplo de Compleción Programable

La forma más común de obtener funcionalidad de compleción adicional más allá de las acciones predeterminadas que proporcionan `complete` y `compgen` es usar una función del intérprete y asociarla a una instrucción particular usando `complete -F`.

La siguiente función proporciona compleciones para la instrucción integrada `cd`. Es un ejemplo razonablemente bueno de lo que las funciones del intérprete deben hacer cuando se usan para compleción. Esta función usa la palabra pasada como `$2` para determinar el nombre de directorio que completar. También puede usar la variable de vector `COMP_WORDS`; la palabra actual es indexada por la variable `COMP_CWORD`.

Esta función cuenta con las instrucciones integradas `complete` y `compgen` para hacer gran parte del trabajo, añadiendo solo las casos que el `cd` de Bash hace más allá de aceptar nombres de directorios básicos: expansión de virgulilla (véase Sección 3.5.2 [Expansión de Virgulilla], página 26), buscar directorios en *$CDPATH*, que se describe anteriormente (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48), y un soporte básico para la opción del intérprete (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73) `cdable_vars`. `_comp_cd` modifica el valor de *IFS* de forma que contiene solo una nueva línea para acomodar los nombres de archivos que contienen espacios y tabulaciones —`compgen` imprime las posibles compleciones que genera una por línea—.

Las posibles compleciones van en la variable de vector *COMPREPLY*, una compleción por elemento de vector. El sistema de compleción programable recoge las compleciones desde ahí cuando la función retorna.

```
# Una función de compleción para la instrucción integrada cd
# basada en la función de compleción de cd del paquete bash_completion
_comp_cd()
{
    local IFS=$' \t\n'    # normaliza IFS
    local cur _skipdot _cdpath
    local i j k

    # Expansión de virgulilla, que también expande
    # la virgulilla al nombre de ruta completo
    case "$2" in
    \~*)    eval cur="$2" ;;
    *)      cur=$2 ;;
    esac

    # ningún cdpath o nombre de ruta absoluto,
    # compleción de directorio directa
    if [[ -z "${CDPATH:-}" ]] || [[ "$cur" == @(./*|../*|/*) ]]; then
        # compgen imprime una ruta por línea;
        # también se podría usar bucle while
        IFS=$'\n'
        COMPREPLY=( $(compgen -d -- "$cur") )
        IFS=$' \t\n'
    # CDPATH+directorios en el directorio actual si no está en CDPATH
```

```
    else
        IFS=$'\n'
        _skipdot=false
        # preprocesa CDPATH para convertir nombres de directorio nulos
        # a .
        _cdpath=${CDPATH/#:/.:}
        _cdpath=${_cdpath//::/:.:}
        _cdpath=${_cdpath/%:/:.}
        for i in ${_cdpath//:/$'\n'}; do
            if [[ $i -ef . ]]; then _skipdot=true; fi
            k="${#COMPREPLY[@]}"
            for j in $( compgen -d -- "$i/$cur" ); do
                COMPREPLY[k++]=${j#$i/}        # Recortar directorio
            done
        done
        $_skipdot || COMPREPLY+=( $(compgen -d -- "$cur") )
        IFS=$' \t\n'
    fi

    # nombres de variable si establecida opción del
    # intérprete apropiada y no compleciones
    if shopt -q cdable_vars && [[ ${#COMPREPLY[@]} -eq 0 ]]; then
        COMPREPLY=( $(compgen -v -- "$cur") )
    fi

    return 0
}
```

Instalamos la función de compleción usando la opción `-F` para `complete`:

```
# Ordena a readline entrecomillar apropiadamente y añadir barras a
# directorios; usa la compleción predeterminada de bash para otros
# argumentos
complete -o filenames -o nospace -o bashdefault -F _comp_cd cd
```

Puesto que nos gustaría que Bash y Readline se hicieran cargo de algunos de los otros detalles por nosotros, usamos otras varias opciones para decirles a Bash y a Readline qué hacer. La opción `-o nombres-de-archivo` le indica a Readline que las posibles compleciones deberían ser tratadas como nombres de archivo, y entrecomilladas apropiadamente. Esta opción también hará que Readline añada una barra a nombres de archivo que pueda reconocer como directorios (por lo cual puede que queramos extender `_comp_cd` para añadir una barra si estamos usando directorios encontrados mediante *CDPATH*: Readline no puede distinguir si esas compleciones son directorios). La opción `-o nospace` le dice a Readline que no añada un carácter de espacio al nombre de directorio, en caso de que queramos agregárselo. La opción `-o bashdefault` trae el resto de las compleciones predeterminadas de Bash —posible compleción que Bash añade al conjunto de Readline habilitado por defecto—. Estas incluyen cosas como la compleción de nombres de instrucciones, compleción de variables para palabras que empiezan por '`{`', compleciones que contienen patrones de expansión de

nombres de archivo (véase Sección 3.5.8 [Expansión de Nombre de Archivo], página 35) y demás.

Una vez instalada usando `complete`, `_comp_cd` será llamado cada vez que intentemos una compleción de palabra para una instrucción `cd`.

Muchos más ejemplos —una extensa colección de compleciones para la mayoría de las instrucciones comunes de GNU, Unix y Linux— están disponibles como parte del proyecto bash_completion. Esto está instalado por defecto en muchas distribuciones de GNU/Linux. Originalmente escrito por Ian Macdonald, el proyecto vive ahora en `http://bash-completion.alioth.debian.org/`. Hay puertos para otros sistemas como Solaris y Mac OS X.

Una versión más antigua del paquete bash_completion se distribuye con bash en el subdirectorio `examples/complete`.

9 Usando el Historial Interactivamente

Este capítulo describe cómo usar la GNU History Library interactivamente, desde el punto de vista de un usuario. Debería ser considerado una guía de usuario. Para información sobre el uso de GNU History Library en otros programas, consulta el GNU Manual de la Readline Library.

9.1 Servicios del Historial de Bash

Cuando la opción `-o history` para la instrucción integrada `set` está habilitada (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68), el intérprete proporciona acceso al *historial de instrucciones*, la lista de instrucciones tecleadas anteriormente. El valor de la variable del intérprete `HISTSIZE` es usada como el número de instrucciones que guardar en una lista del historial. El texto de las `$HISTSIZE` últimas instrucciones (por defecto 500) es guardado. El intérprete almacena cada instrucción en la lista del historial antes de la expansión de parámetros y de variables, pero tras realizar la expansión de historial, sujeta a los valores de las variables del intérprete `HISTIGNORE` y `HISTCONTROL`.

Cuando el intérprete se inicia, el historial es inicializado desde el archivo llamado por la variable `HISTFILE` (por defecto, `~/.bash_history`). El archivo llamado por el valor de `HISTFILE` es cortado, si es necesario, para contener nada más que el número de líneas especificado por el valor de la variable `HISTFILESIZE`. Cuando se cierra un intérprete con el historial habilitado, son copiadas las últimas `$HISTSIZE` líneas desde la lista del historial al archivo nombrado por `$HISTFILE`. Si está establecida (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56) la opción del intérprete `histappend`, las líneas son añadidas al archivo del historial; en caso contrario es sobreescrito el archivo del historial. Si `HISTFILE` está sin establecer, o si el archivo del historial no se puede escribir, no se guarda el historial. Después de guardar el historial, el archivo del historial es cortado para no contener más de `$HISTFILESIZE` líneas. Si `HISTFILESIZE` no está establecido, o está establecido a nulo, o un valor no numérico, o un valor numérico menor que cero, no se corta el archivo del historial.

Si el `HISTTIMEFORMAT` está establecido, la información de marca de tiempo asociada con cada entrada del historial es escrita al archivo del historial, marcada con el carácter de comentario del historial. Cuando el archivo del historial es leído, las líneas que comienzan por el carácter de comentario del historial seguidas inmediatamente por un dígito son interpretadas como marcas de tiempo por la siguiente entrada del historial.

La instrucción integrada `fc` puede ser usada para listar o editar y reejecutar una porción de la lista del historial. La instrucción integrada `history` puede ser usada para mostrar o modificar la lista del historial y manipular el archivo del historial. Al usar la edición en línea de órdenes, están disponibles las instrucciones de búsqueda en cada modo de edición que proporciona acceso a la lista del historial (véase Sección 8.4.2 [Órdenes para el Historial], página 141).

El intérprete proporciona control sobre qué instrucciones son guardadas en la lista del historial. Las variables `HISTCONTROL` y `HISTIGNORE` pueden ser establecidas para hacer que el intérprete solo guarde un subconjunto de las instrucciones introducidas. La opción del intérprete `cmdhist`, si está habilitada, hace que el intérprete trate de guardar cada línea de una instrucción de múltiples líneas en la misma entrada del historial, añadiendo puntos y comas donde sea necesario para preservar la corrección sintáctica. La opción del intérprete

`lithist` hace que el intérprete guarde la instrucción con nuevas líneas intercaladas en vez de puntos y comas. La opción integrada `shopt` es usada para habilitar estas opciones. Véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73, para una descripción de `shopt`.

9.2 Instrucciones Integradas del Historial de Bash

Bash proporciona dos instrucciones integradas que manipulan la lista del historial y el archivo del historial.

`fc`

```
fc [-e nombre-editor] [-lnr] [primera] [última]
fc -s [pat=rep] [instrucción]
```

La primera forma selecciona un rango de instrucciones desde *primera* hasta *última* de la lista del historial y las muestra, o las edita y las vuelve a ejecutar. Tanto *primera* como *última* pueden especificarse como una cadena (para ubicar la instrucción más reciente que comienza con esa cadena) o como un número (un índice de la lista del historial, donde se usa un número negativo como un desplazamiento del número de instrucción actual).

En el listado, una *primera* o *última* de 0 equivale a -1, y -0 equivale a la instrucción actual (normalmente la instrucción `fc`); de lo contrario, 0 equivale a -1, y -0 es inválido.

Si no se especifica *última*, se asigna a *primera*. Si *primera* no se especifica, se asigna a la instrucción anterior para edición y −16 para listado. Si se pasa el argumento `-l`, las instrucciones son listadas en la salida estándar. La opción `-r` invierte el orden del listado. De lo contrario, se llama al editor indicado en *ename* en un archivo que contiene esas instrucciones. Si no se indica *ename*, se usa el valor de la siguiente expansión de variable: `${FCEDIT:-${EDITOR:-vi}}`. Esto quiere decir que use el valor de la variable `FCEDIT` si está asignada, o el valor de la variable `EDITOR` si está asignada, o `vi` si ninguna está asignada. Al completarse la edición, se muestran las instrucciones editadas y se ejecutan.

En la segunda forma, *instrucción* se vuelve a ejecutar después de que cada instancia de *pat* en la instrucción seleccionada es reemplazada por *rep*. *instrucción* se interpreta igual que *primera* arriba.

Un alias útil para usar con la instrucción `fc` es `r='fc -s`, para que teclear '`r cc`' ejecuta la última instrucción que comienza por `cc` y teclear '`r`' vuelva a ejecutar la última instrucción (véase Sección 6.6 [Aliases], página 105).

`history`

```
history [n]
history -c
history -d desplazamiento
history -d inicio-fin
history [-anrw] [nombre-de-archivo]
history -ps arg
```

Sin opciones muestra la lista del historial con números de línea. Las líneas prefijadas con un '`*`' han sido modificadas. Un argumento de *n* lista solo las últimas *n* líneas. Si la variable del intérprete `HISTTIMEFORMAT` está establecida

y no es nula, es usada como una cadena de formato por *strfttime* para mostrar la marca temporal asociada con cada entrada de historial mostrada. Ningún vacío intermedio es mostrado entre la marca temporal formateada y la línea del historial.

Las opciones, si se proporcionan, tienen los siguientes significados:

`-c` Limpia la lista del historial. Esto puede ser combinado con las otras opciones para reemplazar la lista del historial completamente.

`-d` *desplazamiento*
Elimina la entrada del historial en la posición *desplazamiento*. Si *desplazamiento* es positivo, debería ser especificado como si apareciera cuando se muestra el historial. Si *desplazamiento* es negativo, se interpreta como relativo a uno mayor que la última posición del historial, de forma que los índices negativos cuentan hacia atrás desde el final del historial, y un índice de '`-1`' se refiere a la actual instrucción `history -d`.

`-d` *inicio-fin*
Elimina las entradas del historial entre las posiciones *inicio* y *fin*, inclusivas. Los valores positivos y negativos para *start* y *end* se interpretan como se explica arriba.

`-a` Añade las nuevas líneas de historial al archivo del historial. Estas son líneas de historial introducidas desde el comienzo de la actual sesión de Bash, pero no añadidas aún al archivo del historial.

`-n` Añade las líneas del historial no cargadas aún del archivo del historial a la lista actual del historial. Estas son líneas añadidas al archivo del historial desde el comienzo de la actual sesión de Bash.

`-r` Lee el archivo del historial y añade su contenido a la lista del historial.

`-w` Escribe entera la lista del historial actual al archivo del historial.

`-p` Realiza una sustitución de historial en *arg*s y muestra el resultado en la salida estándar, sin guardar los resultados en la lista del historial.

`-s` Los *arg*s son añadidos al final de la lista del historial como una única entrada.

Cuando alguna de las opciones `-w`, `-r`, `-a` o `-n` es usada, si se pasa *nombre-de-archivo*, entonces este es usado como el archivo del historial. Si no, es usado el valor de la variable `HISTFILE`.

9.3 Expansión del Historial

La librería History proporciona una funcionalidad de expansión de historial que es similar a la expansión de historial proporcionada por `csh`. Esta sección describe la sintaxis usada para manipular la información del historial.

Las expansiones del historial introducen palabras desde la lista del historial al flujo de entrada, facilitando repetir instrucciones, insertar los argumentos para una instrucción anterior en la línea de entrada actual o arreglar errores en las instrucciones anteriores rápidamente.

La expansión de historial se realiza inmediatamente después de que sea leída una línea completa, antes de que el intérprete la divida en palabras y se realiza en cada línea de forma individual. Bash trata de informar a las funciones de expansión de historial sobre el antrecomillado aún en efecto de líneas anteriores.

La expansión del historial tiene lugar en dos partes. La primera es para determinar qué línea de la lista del historial debería ser usada durante la sustitución. La segunda es para seleccionar porciones de esa línea para la inclusión en la actual. La línea seleccionada del historial es llamada el *evento*, y las porciones de esa línea sobre las que se actúa son llamadas *palabras*. Varios *modificadores* están disponibles para manipular las palabras seleccionadas. La línea es dividida en palabras de la misma forma que hace Bash, para que varias palabras rodeadas por comillas sean consideradas una palabras. Las expansiones del historial son introducidas por la aparición del carácter de expansión de historial, que es '`!`' por defecto.

La expansión de historial implementa convenciones de entrecomillado del estilo del intérprete: una barra invertida puede ser usada para eliminar el tratamiento especial del siguiente carácter; las comillas simples rodean secuencias literales de caracteres y pueden ser usadas para inhibir la expansión de historial; y los caracteres rodeados por comillas dobles pueden estar sujetos a la expansión de carácter de historial, pero no las comillas dobles, puesto que no son tratadas de forma especial entre comillas dobles.

Al usar el intérprete, solo '\' y ''' pueden ser usados para escapar el carácter de expansión del historial, pero el carácter de expansión del historial también es tratado como entrecomillado si precede inmediatamente a la comillas dobles de cierre en una cadena entrecomillada con comillas dobles.

Varias opciones del intérprete que pueden ser habilitadas con la instrucción integrada `shopt` (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73) pueden ser utilizadas para adaptar el comportamiento de la expansión del historial. Si la opción del intérprete `histverify` está habilitada, y se está usando Readline, las sustituciones del historial no son pasadas inmediatamente al analizador del intérprete. En su lugar, la línea expandida es vuelta a cargar en el búfer de edición de Readline para su posterior modificación. Si se está usando Readline, y la opción del intérprete `histreedit` está habilitada, se volverá a cargar una expansión de historial fallida en el búfer de edición de Readline para su corrección. La opción `-p` para la instrucción integrada `history` puede ser usada para ver qué hará una expansión de historial antes de usarla. La opción `-s` para la instrucción integrada `history` puede ser usada para añadir instrucciones al final de la lista del historial sin realmente ejecutarlas, para que estas estén disponibles para volver a ser llamadas posteriormente. Eso tiene la mayor utilidad en conjunto con Readline.

El intérprete permite el control de varios caracteres usados por el mecanismo de expansión del historial con la variable `histchars`, como se explicó anteriormente (véase Sección 5.2 [Variables de Bash], página 82). El intérprete usa el carácter de comentario del historial para señalar las marcas del tiempo al escribir el archivo del historial.

9.3.1 Designadores de Eventos

Un designador de evento es una referencia a una línea de entrada en la lista del historial. A no ser que la referencia sea absoluta, los eventos son relativos a la actual posición en la lista del historial.

`!` Comienza una sustitución del historial, excepto cuando lo sigue un espacio, tabulación, el final de línea, '`=`' o '`(`' (cuando la opción del intérprete `extglob` es habilitada usando la instrucción integrada `shopt`).

`!`*`n`* Alude a la línea de instrucción *n*.

`!-`*`n`* Alude a la instrucción *n* líneas atrás.

`!!` Alude a la instrucción anterior. Esto es un sinónimo de '`!-1`'.

`!`*`cadena`* Alude a la instrucción más reciente que precede a la actual posición en la lista del historial que comienza por *cadena*.

`!?`*`cadena`*`[?]`
Alude a la instrucción más reciente que precede a la actual posición en la lista del historial que contiene *cadena*. El '`?`' al final puede omitirse si *cadena* es seguida inmediatamente por una nueva línea. Si falta *cadena*, se usa la cadena de la búsqueda más reciente; hay un error si no existe una cadena previa de búsqueda.

`^`*`cadena1`*`^`*`cadena2`*`^`
Sustitución Rápida. Repite la última instrucción, reemplazando *cadena1* por *cadena2*. Equivalente a `!!:s^`*`cadena1`*`^`*`cadena2`*`^`.

`!#` La línea de instrucción completa escrita hasta ahora.

9.3.2 Designadores de Palabras

Los designadores de palabras son usados para seleccionar las palabras deseadas del evento. Un '`:`' separa la especificación de evento del designador de palabra. Puede ser omitido si el designador de palabra empieza por '`^`', '`$`', '`*`', '`-`' o '`%`'. Las palabras son numeradas desde el comienzo de la línea, con la primera palabra indicada por un 0 (cero). Las palabras son insertadas en la línea actual separadas por espacios únicos.

Por ejemplo,

`!!` designa la instrucción precedente. Cuando escribes esto, la instrucción precedente es repetida en su totalidad.

`!!:$` designa el último argumento de la instrucción precedente. Esto puede ser acortado a `!$`.

`!fi:2` designa el segundo argumento de la instrucción más reciente que comienza por las letras `fi`.

Aquí están los designadores de palabras:

`0 (cero)` La palabra número `0`. Para muchas aplicaciones, esto es la palabra de instrucción.

`n` La palabra número *n*.

`^` El primer argumento, es decir, palabra 1.

`$` El último argumento.

`%` La primera palabra que coincide con la búsqueda '`?`*`string`*`?`' más reciente, si la cadena de búsqueda comienza con un carácter que forma parte de una palabra.

`x``-`*`y`* Un rango de palabras; '`-`*`y`*' abrevia a '`0-`*`y`*'.

`*` Todas las palabras, excepto la número `0`. Esto es un sinónimo de '`1-$`'. No es incorrecto usar '`*`' si solo hay una palabra en el evento; la cadena vacía es devuelta en ese caso.

`x``*` Abrevia '*`x`*`-$`'

`x``-` Abrevia '*`x`*`-$`' como '*`x`*`*`', pero omite la última palabra. Si falta '`x`', por defecto es 0.

Si se proporciona un designador de palabra sin una especificación de evento, la anterior instrucción es usada como el evento.

9.3.3 Modificadores

Después del designador de palabra opcional, puede añadir una secuencia de uno o más de los siguientes modificadores, cada uno precedido de un '`:`'. Estos modifican, o editan, la palabra o palabras seleccionadas del historial de eventos.

`h` Elimina un componente de nombre de ruta al final, dejando solo la cabeza.

`t` Elimina todos los componentes de ruta iniciales, dejando la cola.

`r` Elimina un sufijo al final de la forma de '`.`*`sufijo`*', dejando el nombre base.

`e` Elimina todo excepto el sufijo al final.

`p` Imprime la nueva instrucción pero no la ejecuta.

`q` Entrecomilla las palabras sustituidas, escapando sustituciones adicionales.

`x` Cita las palabras sustituidas como con '`q`', pero divide las palabras mediante espacios, tabulaciones y nuevas líneas. Los modificadores '`q`' y '`x`' se excluyen mutuamente; se usa el último proporcionado.

`s/`*`viejo`*`/`*`nuevo`*`/`
Sustituye *nuevo* por la primera ocurrencia de *viejo* en la línea de eventos. Cualquier delimitador puede ser usado en lugar de '`/`'. El delimitador puede ser entrecomillado en *viejo* y *nuevo* con una única barra invertida. Si '`&`' aparece en *nuevo*, es reemplazado por *viejo*. Una única barra invertida entrecomillará el '`&`'. So *viejo* es nulo, se asignará al último *viejo* sustituido o, si no se realizaron sustituciones de historial con anterioridad, la última *cadena* en una búsqueda `!?`*cadena*`[?]`. Si *nuevo* es nulo, se eliminan todas las concidencias de *viejo*. El delimitador final es opcional si es el último carácter en la línea de entrada.

`&` Repite la sustitución anterior.

`g`
`a` Hace que los cambios sean aplicados sobre la línea de evento completa. Usado en conjunto con '`s`', como en `gs/`*`viejo`*`/`*`nuevo`*`/`, o con '`&`'.

`G` Aplica el siguiente modificador '`s`' o '`&`' una vez para cada palabra en el evento.

10 Instalación de Bash

Este capítulo proporciona instrucciones básicas para instalar Bash en las distintas plataformas soportadas. La distribución soporta los sistemas operativos de GNU, casi todas las versiones de Unix y varios sistemas no Unix como BeOS e Interix. Existen otras versiones portadas independientes para MS-DOS, OS/2 y plataformas Windows.

10.1 Instalación Básica

Estas son las instrucciones de instalación para Bash.

La forma más simple de compilar Bash es:

1. Vaya con `cd` al directorio que contiene el código fuente y escriba '`./configure`' para configurar Bash para su sistema. Si está usando `csh` en una vieja versión de System V, puede que necesite escribir '`sh ./configure`' en su lugar para evitar que `csh` intente ejecutar el propio `configure`.

   Ejecutar `configure` toma algo de tiempo. Durante su ejecución, imprime mensajes que dicen qué características está comprobando.
2. Escriba '`make`' para compilar Bash y construir el guion de informe de errores `bashbug`.
3. Opcionalmente, escriba '`make tests`' para ejecutar el conjunto de pruebas de Bash.
4. Escriba '`make install`' para instalar `bash` y `bashbug`. Esto también instalará las páginas de manual y el archivo Info.

El guion del intérprete `configure` trata de adivinar los valores correctos para varias variables dependientes del sistema durante la compilación. Usa esos valores para crear un `Makefile` en cada directorio del paquete (el directorio superior; los directorios `builtins`, `doc` y `support`; cada directorio bajo `lib`; y varios otros). También crea un archivo `config.h` que contiene definiciones dependientes del sistema. Finalmente crea un guion del intérprete llamado `config.status` que puede ejecutar en el futuro para recrear la configuración actual, un archivo `config.cache` que guarda los resultados de sus pruebas para acelerar la reconfiguración y un archivo `config.log` que contiene salida de compilador (útil principalmente para depurar `configure`). Si en algún momento `config.cache` contiene resultados que no quiere conservar, puede eliminarlo o editarlo.

Para averiguar más sobre las opciones y argumentos que el guion `configure` entiende, escriba

```
bash-4.2$ ./configure --help
```

en el prompt de Bash en su directorio de fuentes de Bash.

Si quieres construir Bash en un directorio separado del directorio fuente —para construir para varias arquitecturas, por ejemplo— simplemente usa la ruta completa para el guion configure. Las siguientes instrucciones construirán bash en un directorio bajo `/usr/local/build` desde el código fuente en `/usr/local/src/bash-4.4`:

```
mkdir /usr/local/build/bash-4.4
cd /usr/local/build/bash-4.4
bash /usr/local/src/bash-4.4/configure
make
```

Consulte Sección 10.3 [Compilando para Múltiples Arquitecturas], página 168, para saber más sobre constuir en un directorio separado del código fuente.

Si necesita hacer cosas inusuales para compilar Bash, intente averiguar cómo `configure` podría comprobar si hacerlas o no, y envié diffs o instrucciones a `bash-maintainers@gnu.org` para que puedan ser consideradas para la próxima publicación.

El archivo `configure.ac` se usa para crear `configure` por un programa llamado Autoconf. Solo necesita `configure.ac` si quiere cambiarlo o regenerar `configure` usando una versión más reciente de Autoconf. Si hace esto, asegúrese de que está usando la versión de Autoconf 2.50 o posterior.

Puede eliminar los binarios y archivos objeto del programa del directorio del código fuente escribiendo '`make clean`'. Para eliminar también los archivos que creó `configure` (de forma que pueda compilar Bash para un tipo de ordenador diferente), escriba '`make distclean`'.

10.2 Compiladores y opciones

Algunos sistemas requieren opciones poco usuales para la compilación o el enlazado sobre las que el guion `configure` no sabe. Puede darle a `configure` valores iniciales para variables estableciéndolas en el entorno. Usando un intérprete compatible con Bourne, puede hacer esto en la línea de órdenes así:

```
CC=c89 CFLAGS=-O2 LIBS=-lposix ./configure
```

En sistemas que tienen el programa `env`, puede hacerlo así:

```
env CPPFLAGS=-I/usr/local/include LDFLAGS=-s ./configure
```

El proceso de configuración usa GCC para construir Bash si está disponible.

10.3 Compilando para Múltiples Arquitecturas

Puede compilar Bash para más de un tipo de ordenador al mismo tiempo, ubicando los archivos objeto para cada arquitectura en su propio directorio Para hacer esto, debe usar una versión de `make` que soporte la variable `VPATH`, como GNU `make`. Vaya con `cd` al directorio donde quiere que vayan los archivos objeto y ejecutables y ejecute el guion `configure` desde el directorio fuente (véase Sección 10.1 [Instalación Básica], página 167). Puede que necesite proporcionar el argumento `--srcdir=PATH` para decirle a `configure` dónde están los archivos fuente. `configure` comprueba automáticamente el código fuente en el directorio en que está `cofigure` y en '..'.

Si tiene que usar un `make` que no soporta la variable `VPATH`, puede compilar Bash para una arquitectura a la vez en el directorio del código fuente. Después de que haya instalado Bash para una arquitectura, use '`make distclean`' antes de reconfigurar para otra arquitectura.

Alternativamente, si su sistema soporta enlaces simbólicos, puede usar el guion `support/mkclone` para crear un árbol de construcción que tiene enlaces simbólicos atrás a cada archivo en el directorio de fuentes. He aquí un ejemplo que crea un directorio de construcción en el directorio actual desde un directorio fuente `/usr/gnu/src/bash-2.0`:

```
bash /usr/gnu/src/bash-2.0/support/mkclone -s /usr/gnu/src/bash-2.0 .
```

El guion `mkclone` requiere Bash, así que tiene que tener ya Bash construido para al menos una arquitectura antes de que pueda crear directorios de construcción para otras arquitecturas.

10.4 Nombres de Instalación

Por defecto, '`make install`' instalará en `/usr/local/bin`, `/usr/local/man`, etc. Puede especificar un prefijo de instalación distinto de `/usr/local` pasando a `configure` la opción `--prefix=`*`RUTA`*, o especificando un valor para la variable de '`make`' `DESTDIR` al ejecutar '`make install`'.

Puede especificar prefijos de instalación separados para archivos específicos de arquitectura y archivos independientes de la arquitectura. Si le pasa a `configure` la opción `--exec-prefix=`*`RUTA`*, '`make install`' usará *PATH* como el prefijo para instalar programas y librerías. La documentación y otros archivos de datos todavía usarán el prefijo normal.

10.5 Especificando el Tipo de Sistema

Puede haber algunas funcionalidades que `configure` no puede averiguar automáticamente, pero necesita determinar por el tipo de anfitrión en que se ejecutará Bash. Normalmente `configure` puede averiguarlo, pero si imprime un mensaje diciendo que no puede averiguar el tipo de anfitrión, pásele la opción `--host=TIPO`. '`TIPO`' puede ser un nombre corto para el tipo de sistema, como '`sun4`', o un nombre canónico con tres campos: '`CPU-EMPRESA-SISTEMA`' (p. ej., '`i386-unknown-freebsd4.2`').

Vea el archivo `support/config.sub` para los posibles valores de cada campo.

10.6 Compartiendo Predeterminados

Si quiere asignar valores predeterminados para que los compartan guiones `configure`, puede crear un guion del intérprete del sitio llamado `config.site` que le da valores predeterminados a variables como `CC`, `cache_file` y `prefix`. `configure` mira en `PREFIX/share/config.site` si existe, después en `PREFIX/etc/config.site` si existe. O puede establecer la variable de entorno `CONFIG_SITE` a la ubicación del guion del sitio. Una advertencia: el `configure` de Bash busca un guion de sitio, pero no todos los guiones `configure` lo hacen.

10.7 Controles de Operación

`configure` reconoce las siguientes opciones para controlar cómo opera.

`--cache-file=`*`archivo`*
: Usa y guarda los resultados de las pruebas en *archivo* en vez de `./config.cache`. Asigna `/dev/null` a *archivo* para deshabilitar el caché, para depurar `configure`.

`--help`
: Imprime un resumen de las opciones para `configure`, y finaliza.

`--quiet`
`--silent`
`-q`
: No imprime mensajes diciendo qué comprobaciones se están realizando.

`--srcdir=`*dir*
: Busca el código fuente de Bash en el directorio *dir*. Normalmente `configure` puede averiguar ese directorio automáticamente.

`--version`
: Imprime la versión de Autoconf usada para generar el guion `configure`, y finaliza.

`configure` también acepta alguna otra opción de plantilla, no usadas ampliamente. `configure --help` imprime la lista completa.

10.8 Funcionalidades Opcionales

El `configure` de Bash tiene varias opciones `--enable-`*funcionalidad*, donde *funcionalidad* indica una parte opcional de Bash. También hay varias opciones `--with-`*paquete*, donde *paquete* es algo como '`bash-malloc`' o '`purify`'. Para desactivar el uso predeterminado de un paquete, use `--without-`*paquete*. Para configurar Bash sin una funcionalidad que esté activada por defecto, use `--disable-`*funcionalidad*.

He aquí una lista completa de las opciones `--enable-` y `--with-` que el `configure` de Bash reconoce.

`--with-afs`
: Define si usar el Andrew File System de Transarc.

`--with-bash-malloc`
: Usa la versión de Bash de `malloc` en el directorio `lib/malloc`. Esto no es el mismo `malloc` que aparece en GNU libc, sino una versión más vieja derivada originalmente del `malloc` de BSD 4.2. Este `malloc` es muy rápido, pero desperdicia algo de espacio en cada asignación. Esta opción está activada por defecto. El archivo `NOTES` contiene una lista de sistemas para los cuales esto debería ser desactivado, y `configure` deshabilita esta opción automáticamente para varios sistemas.

`--with-curses`
: Usa la biblioteca curses en vez de la biblioteca termcap. Esto debe ser proporcionado si su sistema tiene una base de datos inadecuada o incompleta de termcap.

`--with-gnu-malloc`
: Un sinónimo para `--with-bash-malloc`.

`--with-installed-readline[=`*PREFIJO*`]`
: Define esto para enlazar Bash con una versión instalada localmente de Readline en vez de la versión en `lib/readline`. Esto solo funciona con las versiones de Readline 5.0 y posteriores. Si *PREFIJO* es `yes` o no proporcionado, `configure` usa los valores de las variables make `includedir` y `libdir`, que son subdirectorios de `prefijo` por defecto, para encontrar la versión instalada de Readline si no está en los directorios estándares del sistema de inclusión y bibliotecas. Si *PREFIJO* es `no`, Bash se enlaza con la versión en `lib/readline`. Si *PREFIJO* está establecido a cualquier otro valor, `configure` lo trata como un nombre de ruta de directorio y busca la versión instalada de Readline en subdirectorios de

ese directorio (archivos de inclusión en *PREFIJO*/`include` y la biblioteca en *PREFIJO*/`lib`).

`--with-purify`
: Define esto para usar el comprobador de ubicación de memoria Purify de Rational Software.

`--enable-minimal-config`
: Esto produce un intérprete con funcionalidades mínimas, cercano al Bourne shell histórico.

Hay varias opciones `--enable-` que alteran cómo se compila y se enlaza Bash, en vez de cambiar funcionalidades de tiempo de ejecución.

`--enable-largefile`
: Habilita soporte para archivos grandes (`http://www.unix.org/version2/whatsnew/lfs20mar.html`) si el sistema operativo requiere opciones especiales de compilador para construir programas que puede acceder a archivos grandes. Esto está habilitado por defecto, si el sistema operativo ofrece soporte para archivos grandes.

`--enable-profiling`
: Esto construye un binario de Bash que produce información de análisis de rendimiento para ser procesada por `gprof` cada vez que se ejecuta.

`--enable-static-link`
: Esto hace que Bash sea enlazado estáticamente, si se usa `gcc`. Esto podría ser usado para construir una versión para usar como intérprete de root.

La opción '`minimal-config`' se puede usar para deshabilitar todas las siguientes opciones, pero se procesa primero, para que se puedan habilitar opciones individuales usando '`enable-`*`funcionalidad`*'.

Todas las siguientes opciones excepto '`disabled-builtins`', '`direxpand-default`' y '`xpg-echo-default`' están habilitadas por defecto, a no ser que el sistema operativo no proporcione el soporte necesario.

`--enable-alias`
: Permite la expansión de alias e incluye las instrucciones integradas `alias` y `unalias` (véase Sección 6.6 [Aliases], página 105).

`--enable-arith-for-command`
: Incluye soporte para la forma alternativa de la instrucción `for` que se comporta como la sentencia `for` del lenguaje C (véase Sección 3.2.5.1 [Construcciones de Bucle], página 11).

`--enable-array-variables`
: Incluye soporte para variables del intérprete de vectores unidimensionales (véase Sección 6.7 [Vectores], página 105).

`--enable-bang-history`
: Incluye soporte para la sustitución de historial estilo `csh` (véase Sección 9.3 [Interacción con el Historial], página 163).

`--enable-brace-expansion`
: Incluye expansión de llaves estilo `csh` (`b{a,b}c` ↦ `bac bbc`). Consulte Sección 3.5.1 [Expansión de Llaves], página 25, para una descripción completa.

`--enable-casemod-attributes`
: Incluye soporte para los atributos que cambian entre mayúsculas y minúsculas en la instrucción integrada `declare` y sentencias de asignación. A las variables con el atributo *uppercase*, por ejemplo, se les convertirán sus valores a mayúscula en la asignación.

`--enable-casemod-expansion`
: Incluye soporte para expansiones de palabras que modifican entre mayúsculas y minúsculas.

`--enable-command-timing`
: Incluye soporte para reconocer `time` como una palabra reservada y para mostrar estadísticas de tiempo para la tubería que sigue a `time` (véase Sección 3.2.3 [Tuberías], página 9). Esto permite que tanto tuberías como instrucciones integradas del intérprete y funciones sean cronometradas.

`--enable-cond-command`
: Incluye soporte para la instrucción condicional `[[`. (véase Sección 3.2.5.2 [Construcciones Condicionales], página 12).

`--enable-cond-regexp`
: Incluye soporte para coincidir expresiones regulares POSIX usando el operador binario '`=~`' en la instrucción condicional `[[`. (véase Sección 3.2.5.2 [Construcciones Condicionales], página 12).

`--enable-coprocesses`
: Incluye soporte para coprocesos y la palabra reservada `coproc` (véase Sección 3.2.3 [Tuberías], página 9).

`--enable-debugger`
: Incluye soporte para el depurador de Bash (distribuido por separado).

`--enable-dev-fd-stat-broken`
: Si ejecutar `stat` en /dev/fd/*N* devuelve diferentes resultados que ejecutar `fstat` en el descriptor de archivo *N*, proporciona esta opción para activar una solución alternativa. Esto tiene implicaciones para instrucciones condicionales que comprueban los atributos de los archivos.

`--enable-direxpand-default`
: Hace que la opción del intérprete `direxpand` (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73) sea habilitada por defecto cuando se inicia el intérprete. Normalmente está deshabilitada por defecto.

`--enable-directory-stack`
: Incluye soporte para una pila de directorios del estilo `csh` y las instrucciones integradas `pushd`, `popd` y `dirs` (véase Sección 6.8 [La Pila de Directorios], página 107).

`--enable-disabled-builtins`
: Permite que las instrucciones integradas sean invocadas mediante '`builtin xxx`' incluso después de que `xxx` haya sido deshabilitado usando '`enable -n xxx`'. Consulte Sección 4.2 [Instrucciones Integradas de Bash], página 56, para detalles de las instrucciones integradas `builtin` y `enable`.

`--enable-dparen-arithmetic`
: Incluye soporte para la instrucción `((...))` (véase Sección 3.2.5.2 [Construcciones Condicionales], página 12).

`--enable-extended-glob`
: Incluye soporte para las funcionalidades extendidas de coincidencia de patrones descritas antes en Sección 3.5.8.1 [Coincidencia de Patrones], página 36.

`--enable-extended-glob-default`
: Habilita la opción predeterminada de la opción del intérprete *extglob* descrita anteriormente en Sección 4.3.2 [La Instrucción Integrada Shopt], página 73.

`--enable-function-import`
: Incluye soporte para importar definiciones de función exportadas por otra instancia del intérprete desde el entorno. Esta opción está habilitada por defecto.

`--enable-glob-asciirange-default`
: Habilita el valor predeterminado de la opción del intérprete *globasciiranges* descrita antes en Sección 4.3.2 [La Instrucción Integrada Shopt], página 73. Esto controla el comportamiento de rangos de caracteres cuando se usa en expresiones de llaves de coincidencia de patrones.

`--enable-help-builtin`
: Incluye la instrucción integrada `help`, que muestra ayuda sobre las instrucciones integradas y variables del intérprete (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).

`--enable-history`
: Incluye el historial de instrucciones y las instrucciones integradas `fc` y `history` (véase Sección 9.1 [Servicios del Historial de Bash], página 161).

`--enable-job-control`
: Esto habilita las funcionalidades de control de tareas (véase Capítulo 7 [Control de Tareas], página 118), si el sistema operativo las soporta.

`--enable-multibyte`
: Esto habilita el soporte para caracteres multibyte si el sistema operativo proporciona el soporte necesario.

`--enable-net-redirections`
: Esto habilita el manejo especial de nombres de archivo de la forma `/dev/tcp/`*`anfitrión`*`/`*`puerto`* y `/dev/udp/`*`host`*`/`*`port`* cuando se usan en redirecciones (véase Sección 3.6 [Redirecciones], página 37).

`--enable-process-substitution`
: Esto permite la sustitución de procesos (véase Sección 3.5.6 [Sustitución de Procesos], página 34) si el sistema operativo proporciona el soporte necesario.

`--enable-progcomp`
: Habilita las facilidades de compleción programable (véase Sección 8.6 [Compleción Programable], página 150). Si Readline no está habilitado, esta opción no tiene efecto.

`--enable-prompt-string-decoding`
: Activa la interpretación de varios caracteres escapados con barras invertidas en las cadenas del prompt `$PS0`, `$PS1`, `$PS2` y `$PS4`. Consulte Sección 6.9 [Controlando el Prompt], página 109, para una lista completa de secuencias de escape de prompt.

`--enable-readline`
: Incluye soporte para edición en línea de órdenes e historial con la versión de Bash de la biblioteca Readline (véase Capítulo 8 [Edición en Línea de Órdenes], página 123).

`--enable-restricted`
: Incluye soporte para un *intérprete restringido*. Si esto está habilitado, Bash, cuando se ejecuta como `rbash`, entra en un modo restringido. Consulte Sección 6.10 [El Intérprete Restringido], página 110, para una descripción del modo restringido.

`--enable-select`
: Incluye la instrucción compuesta `select`, que permite la generación de menús simples (véase Sección 3.2.5.2 [Construcciones Condicionales], página 12).

`--enable-separate-helpfiles`
: Usa archivos externos para la documentación mostrada por la instrucción integrada `help` en vez de almacenar el texto internamente.

`--enable-single-help-strings`
: Guarda el texto mostrado por la instrucción integrada `help` como una cadena simple para cada tema de ayuda. Esto ayuda a traducir el texto a diferentes idiomas. Puede que necesite deshabilitar esto si su compilador no puede manejar literales de cadenas muy largos.

`--enable-strict-posix-default`
: Hace a Bash conforme con el estándar POSIX por defecto (véase Sección 6.11 [Modo POSIX de Bash], página 111).

`--enable-usg-echo-default`
: Un sinónimo para `--enable-xpg-echo-default`.

`--enable-xpg-echo-default`
: Hace que la instrucción integrada `echo` expanda los caracteres escapados por barras invertidas por defecto, sin requerir la opción `-e`. Esto establece el valor predeterminado de la opción del intérprete `xpg_echo` a `on`, que hace que el `echo` de Bash se comporte más como la versión especificada en la Especificación Única de Unix, versión 3. Véase Sección 4.2 [Instrucciones Integradas de Bash], página 56, para una descripción de las secuencias de escape que `echo` reconoce.

El archivo `config-top.h` contiene sentencias de Preprocesador C '`#define`' para opciones que no se pueden establecer desde `configure`. Algunas de estas no están pensadas para

cambiarse; aténgase a las consecuencias si lo hace. Lea los comentarios asociados con cada definición para más información sobre su efecto.

Apéndice A Notificar Errores

Por favor, reporte todos los errores que encuentre en Bash. Pero primero, debería asegurarse de que es realmente un error, y de que aparece en la última versión de Bash. La última versión de Bash siempre está disponible para FTP desde `ftp://ftp.gnu.org/pub/gnu/bash/`.

Una vez que haya determinado que existe realmente un error, use la instrucción `bashbug` para enviar un informe de fallo. Si tiene un arreglo, le animamos a que nos envíe eso también! Sugerencias e informes de errores 'filosóficos' pueden ser enviados a `bug-bash@gnu.org` o publicados en el grupo de noticias Usenet `gnu.bash.org`.

Todos los informes de fallo deberían incluir:

- El número de versión de Bash.
- El hardware y el sistema operativo.
- El compilador usado para compilar Bash.
- Una descripción del comportamiento del error.
- Un guion corto o 'receta' que produzca el fallo y pueda ser usado para reproducirlo.

`bashbug` inserta los primeros tres elementos automáticamente en la plantilla que proporciona para enviar un informe de fallo.

Por favor, envíe todos los informes relativos a este manual a `bug-bash@gnu.org`.

Apéndice B Diferencias Principales Respecto a The Bourne Shell

Bash implementa esencialmente la misma gramática, expansión de parámetros y variables, redirección y entrecomillado que el Bourne Shell. Bash usa el estándar POSIX como la especificación de cómo deberían ser implementadas estas características. Hay algunas diferencias entre el intérprete Bourne tradicional y Bash; esta sección detalla rápidamente las diferencias significantes. Varias de estas diferencias se explican detalladamente en secciones previas. Esta sección usa la versión de `sh` incluida en SVR4.2 (la última versión del histórico intérprete Bourne) como la referencia base.

- Bash está conforme con POSIX incluso cuando la especificación POSIX difiere del comportamiento tradicional de `sh` (véase Sección 6.11 [Modo POSIX de Bash], página 111).
- Bash tiene opciones de llamada de varios caracteres (véase Sección 6.1 [Llamando a Bash], página 95).
- Bash tiene edición en línea de órdenes (véase Capítulo 8 [Edición en Línea de Órdenes], página 123) y la instrucción integrada `bind`.
- Bash proporciona un mecanismo de compleción programable de palabra (véase Sección 8.6 [Compleción Programable], página 150), y las instrucciones integradas `complete`, `compgen` y `compopt` para manipularlo.
- Bash tiene historial de instrucciones (véase Sección 9.1 [Servicios del Historial de Bash], página 161) y las instrucciones integradas `history` y `fc` para manipularlo. La lista del historial de Bash mantiene información de marcas de tiempo y usa el valor de la variable `HISTTIMEFORMAT` para mostrarla.
- Bash implementa expansión del historial estilo `csh` (véase Sección 9.3 [Interacción con el Historial], página 163).
- Bash tiene variables de vectores unidimensionales (véase Sección 6.7 [Vectores], página 105), y las apropiadas expansiones de variable y sintaxis de asignación para usarlos. Varias de las instrucciones integradas de Bash toman opciones para actuar sobre vectores. Bash ofrece una serie de variables de vectores integradas.
- La sintaxis de entrecomillado `$'...'`, que expande caracteres de barra invertida ANSI-C en el texto entre las comillas simples, está soportada (véase Sección 3.1.2.4 [Entrecomillado ANSI-C], página 7).
- Bash soporta la sintaxis de entrecomillado `$"..."` para hacer tranducciones de región específica de los caracteres entre las comillas dobles. Las opciones de llamada `-D`, `--dump-strings` y `--dump-po-strings` listan las cadenas convertibles encontradas en un guion (véase Sección 3.1.2.5 [Traducción de Localización], página 7).
- Bash implementa la palabra clave `!` para negar el valor de retorno de una tubería (véase Sección 3.2.3 [Tuberías], página 9). Muy útil cuando una sentencia `if` solo tiene que actuar si una comprobación falla. La opción de Bash '`-o pipefail`' para `set` hará que una tubería devuelva un estado de fallo si falla cualquier instrucción.
- Bash tiene la palabra reservada `time` y el cronometrado de instrucciones (véase Sección 3.2.3 [Tuberías], página 9). La muestra de las estádisticas de tiempo puede ser controlada por la variable `TIMEFORMAT`.
- Bash implementa la aritmética `for (( expr1 ; expr2 ; expr3 ))` para instrucción, similar al lenguaje C (véase Sección 3.2.5.1 [Construcciones de Bucle], página 11).

- Bash incluye la instrucción compuesta `select`, que permite la generación de menús simples (véase Sección 3.2.5.2 [Construcciones Condicionales], página 12).
- Bash incluye la instrucción compuesta `[[`, que hace la comprobación condicional parte de la gramática del intérprete (véase Sección 3.2.5.2 [Construcciones Condicionales], página 12), incluyendo correspondencias opcionales de expresiones regulares.
- Bash proporciona coincidencia opcional independiente de minúsculas y mayúsculas para las construcciones `case` y `[[`.
- Bash incluye expansión de llaves (véase Sección 3.5.1 [Expansión de Llaves], página 25) y expansión de virgulilla (véase Sección 3.5.2 [Expansión de Virgulilla], página 26).
- Bash implementa aliases de instrucciones y las instrucciones integradas `alias` y `unalias` (véase Sección 6.6 [Aliases], página 105).
- Bash proporciona aritmética del intérprete, la instrucción compuesta `((` (véase Sección 3.2.5.2 [Construcciones Condicionales], página 12) y expansión aritmética (véase Sección 6.5 [Aritmética del Intérprete], página 103).
- Las variables presentes en el entorno inicial del intérprete son exportadas automáticamente a procesos hijos. El Bourne shell no hace esto normalmente a no ser que las variables sean explícitamente usando la instrucción `export`.
- Bash soporta el operador de asignación '`+=`', que añade al valor de la variable nombrada en el lado izquierdo.
- Bash incluye las expansiones de eliminación de patrones POSIX '`%`', '`#`', '`%%`' y '`##`' para eliminar subcadenas iniciales o finales de valores de variables (véase Sección 3.5.3 [Expansión de Parámetros del Intérprete], página 27).
- La expansión `${#xx}`, que devuelve la longitud de `${xx}`, está soportada (véase Sección 3.5.3 [Expansión de Parámetros del Intérprete], página 27).
- La expansión `${var:`*desplazamiento*`[:`*longitud*`]}`, que se expande a la subcadena del valor de `var` de longitud *longitud*, empezando en *desplazamiento*, está presente (véase Sección 3.5.3 [Expansión de Parámetros del Intérprete], página 27).
- La expansión `${var/[/]`*patrón*`[/`*reemplazo*`]}`, que coincide con *patrón* y lo reemplaza por *reemplazo* en el valor de `var`, está disponible (véase Sección 3.5.3 [Expansión de Parámetros del Intérprete], página 27).
- La expansión `${!`*prefijo*`*}`, que se expande a los nombres de todas las variables del intérprete cuyos nombres comienzan por *prefijo*, está disponible (véase Sección 3.5.3 [Expansión de Parámetros del Intérprete], página 27).
- Bash tiene expansión de variable *indirecta* usando `${!palabra}` (véase Sección 3.5.3 [Expansión de Parámetros del Intérprete], página 27).
- Bash puede expandir parámetros posicionales más allá de `$9` usando `${`*num*`}`.
- Se implementa la forma POSIX `$()` de sustitución de instrucciones (véase Sección 3.5.4 [Sustitución de Instrucciones], página 33), y preferida al `‘‘` del Bourne shell (que también se implementa para compatibilidad hacia atrás).
- Bash tiene sustitución de procesos (véase Sección 3.5.6 [Sustitución de Procesos], página 34).
- Bash automáticamente asigna variables que proporcionan información sobre el usuario actual (`UID`, `EUID` y `GROUPS`), el anfitrión actual (`HOSTTYPE`, `OSTYPE`, `MACHTYPE` y

`HOSTNAME`), y la instancia de Bash que se está ejecutando (`BASH`, `BASH_VERSION`, and `BASH_VERSINFO`). Véase Sección 5.2 [Variables de Bash], página 82, para detalles.

- La variable `IFS` es usada para dividir solo los resultados de expansión, no todas las palabras (véase Sección 3.5.7 [División de Palabras], página 34). Esto cierra un antiguo agujero de seguridad del intérprete.
- El código de la expansión de nombre de archivo usa '`!`' y '`^`' para negar el conjunto de caracteres entre las llaves. El Bourne shell usa solo '`!`'.
- Bash implementa el conjunto completo de operadores POSIX de expansión de nombre de archivo, incluyendo *clases de caracteres*, *clases de equivalencia* y *símbolos de ordenación* (véase Sección 3.5.8 [Expansión de Nombre de Archivo], página 35).
- Bash implementa funcionalidades extendidas de coincidencia de patrones cuando está habilitada la opción del intérprete `extglob` (véase Sección 3.5.8.1 [Coincidencia de Patrones], página 36).
- Es posible tener una variable y una función con el mismo nombre; `sh` no separa los dos espacios de nombre.
- Se permite a las funciones de Bash tener variables locales usando la instrucción integrada `local`, y así se pueden escribir funciones recursivas útiles (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).
- Las asignaciones de variables que preceden a instrucciones afectan solo a esa instrucción, incluso instrucciones intérgradas y funciones (véase Sección 3.7.4 [Entorno], página 44). En `sh`, todas las asignaciones de variables que preceden a instrucciones son globales a no ser que la instrucción se ejecute desde el sistema de archivos.
- Bash realiza expansión de nombre de archivo en nombres de archivos especificados como operandos para operadores de redirección de entrada y salida (véase Sección 3.6 [Redirecciones], página 37).
- Bash contiene el operador de redirección '`<>`', permitiendo a un archivo ser abierto tanto para lectura como escritura, y el operador de redirección '`&>`', para dirigir salida estándar y error estándar al mismo archivo (véase Sección 3.6 [Redirecciones], página 37).
- Bash incluye el operador de redirección '`<<<`', permitiendo que una cadena sea usada como la entrada estándar en una instrucción.
- Bash implementa los operadores de redirección '`[n]<&`*palabra*' y '`[n]>&`*palabra*', que mueven un descriptor de archivo a otro.
- Bash trata varios nombres de archivo de forma especial cuando son usados en operadores de redirección (véase Sección 3.6 [Redirecciones], página 37).
- Bash puede abrir conexiones de red a máquinas y servicios arbitrarios con los operadores de redirección (véase Sección 3.6 [Redirecciones], página 37).
- La opción `noclobber` está disponible para evitar sobrescribir archivos existentes con la redirección de salida (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68). El operador de redirección '`>|`' puede ser usado para sobrescribir `noclobber`.
- Las instrucciones integradas de Bash `cd` y `pwd` (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48) toman las opciones `-L` y `-P` para alternar entre los modos lógico y físico.

- Bash permite a una función sobrescribir una instrucción integrada con el mismo nombre y proporciona acceso a la funcionalidad de esa instrucción integrada dentro de la función a través de las instrucciones integradas `builtin` y `command` (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).
- La instrucción integrada `command` permite deshabilitar selectivamente funciones cuando se realiza la búsqueda de instrucciones (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).
- Se pueden habilitar o deshabilitar instrucciones integradas individuales usando la instrucción integrada `enable` (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).
- La instrucción integrada de Bash `exec` toma opciones adicionales que permiten a los usuarios controlar los contenidos pasados a la instrucción ejecutada, y cuál va a ser el argumento número 0 (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48).
- Las funciones del intérprete pueden ser exportadas a hijos a través del entorno usando `export -f` (véase Sección 3.3 [Funciones del Intérprete], página 19).
- Las instrucciones integradas de Bash `export`, `readonly` y `declare` pueden tomar una opción `-f` para actuar en funciones del intérprete, una opción `-p` para mostrar variables con varios atributos asignados en un formato que puede ser reusado como entrada del intérprete, una opción `-n` para eliminar varios atributos de variable y argumentos '`nombre=valor`' para establecer atributos y valores de variable simultáneamente.
- La instrucción integrada `hash` de Bash permite que un nombre sea asociado con un nombre de archivo arbitrario, incluso cuando ese nombre de archivo no pueda ser encontrado buscando el `$PATH`, usando '`hash -p`' (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48).
- Bash incluye una instrucción integrada `help` como referencia rápida a facilidades del intérprete (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).
- La instrucción integrada `printf` está disponible para mostrar salida con formato (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).
- La instrucción integrada de Bash `read` (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56) leerá una línea acabada en '\' con la opción `-r`, y usará la variable `REPLY` como predeterminada si no se proporciona ningún argumento que no sea una opción. La instrucción integrada de Bash `read` también acepta una cadena de prompt con la opción `-p` y usará Readline para obtener la línea cuando se pase la opción `-e`. La instrucción integrada `read` también tiene opciones adicionales para controlar la entrada: la opción `-s` desactivará la impresión de caracteres cuando son leídos, la opción `-t` permitirá a `read` quedarse sin tiempo si la entrada no llega dentro de un número de segundos especificado, la opción `-n` permitirá leer solo un número especificado de caracteres en vez de una línea completa y la opción `-d` leerá hasta un carácter particular en vez de hasta nueva línea.
- La instrucción integrada `return` puede ser usada para abortar la ejecución de guiones ejecutados con las instrucciones integradas `.` o `source` (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48).
- Bash incluye la instrucción integrada `shopt`, para mejor control de las capacidades opcionales de Bash (véase Sección 4.3.2 [La Instrucción Integrada Shopt], página 73), y

permite habilitar y deshabilitar estas opciones durante la llamada del intérprete (véase Sección 6.1 [Llamando a Bash], página 95).

- Bash tiene un comportamiento mucho más opcional controlable con la instrucción integrada `set` (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68).
- La opción '`-x`' (`xtrace`) muestra instrucciones aparte de instrucciones simples al realizar un rastreo de ejecución (véase Sección 4.3.1 [La Instrucción Integrada Set], página 68).
- La instrucción integrada `test` (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56) es ligeramente diferente, a como implementa el algoritmo POSIX, que especifica el comportamiento basado en el número de argumentos.
- Bash incluye la instrucción integrada `caller`, que muestra el contexto de cualquier llamada de subrutina activa (una función del intérprete o un guion ejecutado con las instrucciones integradas `.` o `source`). Esto soporta el depurador de Bash.
- La instrucción integrada `trap` (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48) permite una especificación de pseudoseñal `DEBUG`, similar a `EXIT`. Las instrucciones especificadas con una trap `DEBUG` se ejecutan antes de cada instrucción simple, instrucción `for`, instrucción `case`, instrucción `select`, cada instrucción aritmética `for`, y antes de que la primera instrucción se ejecute en una función del intérprete. La trap `DEBUG` no es heredada por funciones del intérprete a no ser que se haya dado a la función el atributo `trace` o la opción `functrace` haya sido habilitada usando la instrucción integrada `shopt`. La opción del intérprete `extdebug` tiene efectos adicionales en la trap `DEBUG`.

  La instrucción integrada `trap` (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48) permite una especificación de pseudoseñal `ERR`, similar a `EXIT` y `DEBUG`. Las instrucciones especificadas con una trap `ERR` se ejecutan después de que falle una instrucción simple, con unas pocas excepciones. La trap `ERR` no es heredada por funciones del intérprete a no ser que esté habilitada la opción `-o errtrace` para la instrucción integrada `set`.

  La instrucción integrada `trap` (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48) permite una especificación de pseudoseñal `RETURN`, similar a `EXIT` y `DEBUG`. Las instrucciones especificadas con una trap `RETURN` son ejecutadas antes de que la ejecución se reanude después de que una función del intérprete o un guion del intérprete se ejecute con retornos `.` o `source`. La trap `RETURN` no se hereda por funciones del intérprete a no ser que se haya dado a la función el atributo `trace` o la opción `functrace` haya sido habilitada usando la instrucción integrada `shopt`.
- La instrucción integrada `type` de Bash es más extensa y da más información sobre los nombres que encuentra (véase Sección 4.2 [Instrucciones Integradas de Bash], página 56).
- La instrucción integrada `umask` de Bash permite que una opción `-p` haga que la salida sea mostrada en la forma de la instrucción `umask` que pueda ser reusada como entrada (véase Sección 4.1 [Instrucciones Integradas del Bourne Shell], página 48).
- Bash implementa una pila de directorios del estilo `csh` y proporciona las instrucciones `pushd`, `popd` y `dirs` para manipularla (véase Sección 6.8 [La Pila de Directorios], página 107). Bash hace también visible la pila de directorios como el valor de la variable del intérprete `DIRSTACK`.

- Bash interpreta caracteres especiales escapados con barras invertidas en las cadenas del prompt cuando es interactivo (véase Sección 6.9 [Controlando el Prompt], página 109).
- El modo restringido de Bash es más útil (véase Sección 6.10 [El Intérprete Restringido], página 110); el modo restringido del intérprete SVR4.2 es demasiado restringido.
- La instrucción integrada `disown` puede eliminar una tarea de la tabla interna de tareas del intérprete (véase Sección 7.2 [Instrucciones Integradas de Control de Tareas], página 119) o suprimir el envío de `SIGHUP` a una tarea cuando finaliza el intérprete como el resultado de una `SIGHUB`.
- Bash incluye varias funcionalidades para ofrecer un depurador separado para guiones del intérprete.
- El intérprete SVR4.2 tiene dos instrucciones integradas relacionadas con privilegios (`mldmode` y `priv`) inexistentes en Bash.
- Bash no tiene las instrucciones integradas `stop` o `newgrp`.
- Bash no usa la variable `SHACCT` o realiza contabilidad del intérprete.
- El `sh` SVR4.2 usa una variable `TIMEOUT` como Bash usas `TMOUT`.

Se pueden encontrar más funcionalidades de Bash en Capítulo 6 [Funcionalidades de Bash], página 95.

B.1 Diferencias de Implementación Respecto al Intérprete SVR4.2

Puesto que Bash es una implementación completamente nueva, no tiene muchas de las limitaciones del intérprete SVR4.2. Por ejemplo:

- Bash no se bifurca en un subintérprete al redirigir en o desde una estructura de control como una sentencia `if` o `while`.
- Bash no permite comillas sin correspondencia. El intérprete SVR4.2 insertará silenciosamente una comilla de cierre necesaria en `EOF` bajo ciertas circunstancias. Esto puede ser la causa de algunos errores difíciles de encontrar.
- El intérprete SVR4.2 usa un pomposo esquema de gestión de memoria basado en el atrapado de `SIGSEGV`. Si se inicia el intérprete desde un proceso con `SIGSEGV` bloqueado (p. ej., usando la llamada de la biblioteca C `system()`), actúa mal.
- En un cuestionable intento de seguridad, el intérprete SVR4.2, al llamarse sin la opción `-p`, alterará sus UID y GID reales y efectivos si son menores que algún umbral mágico, comúnmente 100. Esto puede llevar a resultados inesperados.
- El intérprete SVR4.2 no permite a usuarios usar las traps `SIGSEGV`, `SIGALRM` o `SIGCHLD`.
- El intérprete SVR4.2 no permite eliminar las variables `IFS`, `MAILCHECK`, `PATH`, `PS1` o `PS2`.
- El intérprete SVR4.2 trata '`^`' como el equivalente sin documentar de '`|`'.
- Bash permite múltiples argumentos de opción cuando es llamado (`-x -v`); el intérprete SVR4.2 permite solo un argumento de opción (`-xv`). De hecho, algunas versiones del intérprete vuelcan la memoria si el segundo argumento empieza con un '`-`'.
- El intérprete SVR4.2 finaliza un guion si falla cualquier instrucción integrada; Bash finaliza un guion solo si una de las instrucciones integradas especiales POSIX falla, y solo para ciertos fallos, según se enumera en el estándar POSIX.

- El intérprete SVR4.2 se comporta diferente cuando se llama como `jsh` (activa el control de tareas).

Apéndice C GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Apéndice D Traducción de GNU Free Documentation License

Versión 1.3, 3 de noviembre de 2008

This is an unofficial translation of the GNU Free Documentation License, version 1.3 into Spanish. It was not published by the Free Software Foundation, and does not legally state the distribution terms for software that uses the GNU FDL—only the original English text of the GNU FDL does that. However, we hope that this translation will help language speakers understand the GNU FDL better.

Esta es una traducción no oficial de la GNU Free Documentation License, versión 1.3 al español. No fue publicada por la Free Software Foundation, y no establece legalmente los términos de distribución para software que usa la GNU FDL —solo el texto original de la GNU FDL hace eso—. Sin embargo, esperamos que esta traducción ayude a hablantes de español a entender la GNU FDL mejor.

Puede publicar esta traducción, modificada o sin modificar bajo los términos en `http://www.gnu.org/licenses/translations.html`.

[Comienzo de la traducción de la licencia]
Se permite a cualquier persona copiar y distribuir copias literales de este documento, pero cambiarlo no está permitido.

0. PREÁMBULO

El propósito de esta Licencia es permitir que un manual, libro de texto u otro documento funcional y útil sea *libre* en el sentido de libertad: asegurar a todo el mundo la libertad efectiva de copiarlo y redistribuirlo, con o sin modificaciones, de manera comercial o no. En segundo término, esta Licencia reserva al autor y al editor de una manera de obtener reconocimiento por su trabajo, sin que se le considere responsable de las modificaciones realizadas por otros.

Esta Licencia es un tipo de «copyleft», lo que significa que los trabajos derivados del documento deben a su vez ser libres en el mismo sentido. Complementa a la GNU General Public License, que es una licencia copyleft diseñada para el software libre.

Hemos diseñado esta Licencia para usarla en manuales de software libre, ya que el software libre necesita documentación libre: un programa libre debe venir con manuales que ofrezcan la mismas libertades que el software. Pero esta Licencia no se limita a manuales de software; puede usarse para cualquier obra textual, sin tener en cuenta su temática o si se publica como libro impreso o no. Recomendamos esta licencia principalmente para trabajos cuyo fin sea instructivo o de referencia.

1. DEFINICIONES Y APLICABILIDAD

Esta Licencia se aplica a cualquier manual u otro trabajo, en cualquier soporte, que contenga una aviso puesto por el propietario de los derechos de autor que indique que puede ser distribuido bajo los términos de esta Licencia. Tal aviso garantiza en

cualquier lugar del mundo, sin pago de derechos y sin límite de tiempo, el uso de dicho trabajo según las condiciones aquí estipuladas. En adelante la palabra «Documento» se referirá a cualquiera de dichos manuales o trabajos. Cualquier persona del público es una licenciataria y será referida como Usted. Usted acepta la licencia si copia, modifica o distribuye el trabajo de cualquier modo que requiera permiso según la ley de propiedad intelectual.

Una «Versión Modificada» del Documento significa que cualquier trabajo que contenga el Documento o una porción del mismo, ya sea una copia literal o con modificaciones y/o traducciones a otro idioma.

Una «Sección Secundaria» es un apéndice con título o una sección preliminar del Documento que trata exclusivamente de la relación entre los autores o editores del Documento con el tema general del Documento (o temas relacionados) pero que no contiene nada que entre directamente en dicho tema general. (Por ejemplo, si el Documento es en parte un libro de texto de matemáticas, una Sección Secundaria puede no explicar nada de matemáticas). La relación podría ser un asunto de conexión histórica con el tema o temas relacionados, o de opinión legal, comercial, filosófica, ética o política acerca de ellos.

Las «Secciones Invariantes» son ciertas Secciones Secundarias cuyos títulos son designados, como aquellos de Secciones Invariantes, en el aviso que indica que el documento está publicado bajo esta Licencia. Si una sección no entra en la definición de Secundaria, no puede designarse como Invariante. El documento puede no tener Secciones Invariantes. Si el Documento no identifica las Secciones Invariantes, es que no las tiene.

Los «Textos de Cubierta» son ciertos pasajes cortos de texto que se listan, como Textos de Cubierta Delantera o Textos de Cubierta Trasera, en el aviso que indica que el documento está publicado bajo esta Licencia. Un Texto de Cubierta Delantera puede tener como mucho 5 palabras, y uno de Cubierta Trasera puede tener hasta 25 palabras.

Una copia «Transparente» del Documento, significa una copia para lectura de máquina, representada en un formato cuya especificación está disponible al público general, apta para que los contenidos puedan ser revisados directamente con editores de texto genéricos o (para imágenes compuestas de píxeles) con programas genéricos de manipulación de imágenes o (para dibujos) con algún editor de dibujos ampliamente disponible, y que sea adecuado como entrada para formateadores de texto o para su traducción automática a formatos adecuados para formateadores de texto. Una copia hecha en un formato definido como Transparente, pero cuyo marcaje o ausencia de él haya sido diseñado para impedir o dificultar modificaciones posteriores por parte de los lectores no es Transparente. Un formato de imagen no es transparente si se usa para una cantidad de texto sustancial. Una copia que no es «Transparente» se denomina Opaca.

Ejemplos de formatos adecuados para copias Transparentes son ASCII puro sin marcaje, formato de entrada de Texinfo, formato de entrada de LaTeX, SGML o XML usando una DTD disponible públicamente, y HTML, PostScript o PDF simples, que sigan los estándares y diseñados para que los modifiquen personas. Ejemplos de formatos de imagen transparentes son PNG, XCF y JPG. Los formatos Opacos incluyen formatos propietarios que pueden ser leídos y editados únicamente en procesadores de palabras propietarios, SGML o XML para los cuáles las DTD y/o herramientas de procesamiento no estén ampliamente disponibles, y HTML, PostScript o PDF generados por algunos procesadores de palabras solo como salida.

La «Página de Título» significa, en un libro impreso, la página de título, más las páginas siguientes que sean necesarias para mantener legiblemente el material que esta Licencia requiere en la portada. Para trabajos en formatos que no tienen página de portada como tal, «Portada» significa el texto cercano a la aparición más prominente del título del trabajo, precediendo el comienzo del cuerpo del texto.

El «Editor» se refiere a cualquier persona o entidad que distribuya copias del Documento a el público.

Una sección «Titulada XYZ» significa una parte del Documento cuyo título es precisamente XYZ o contiene XYZ entre paréntesis, a continuación de texto que traduce XYZ a otro idioma. (Aquí XYZ se refiere a nombres de sección específicos mencionados más abajo, como «Agradecimientos», «Dedicatorias», «Aprobaciones» o «Historia».) «Conservar el Título» de tal sección cuando se modifica el Documento significa que permanece una sección «Titulada XYZ» según esta definición.

El Documento puede incluir Limitaciones de Garantía cercanas al aviso donde se declara que al Documento se le aplica esta Licencia. Se considera que estas Limitaciones de Garantía están incluidas, por referencia, en la Licencia, pero solo en cuanto a limitaciones de garantía: cualquier otra implicación que estas Limitaciones de Garantía puedan tener es nula y no tiene efecto en el significado de esta Licencia.

2. COPIA LITERAL

   Usted puede copiar y distribuir el Documento en cualquier soporte, sea en forma comercial o no, siempre y cuando esta Licencia, los avisos de copyright y la nota que indica que esta Licencia se aplica al Documento se reproduzcan en todas las copias y que usted no añada ninguna otra condición a las expuestas en esta Licencia. Usted no puede usar medidas técnicas para obstruir o controlar la lectura o copia posterior de las copias que usted haga o distribuya. Sin embargo, usted puede aceptar compensación a cambio de las copias. Si distribuye un número suficientemente grande de copias también deberá seguir las condiciones de la sección 3.

   Usted también puede prestar copias, bajo las mismas condiciones establecidas anteriormente, y puede exhibir copias públicamente.

3. COPIAR EN CANTIDAD

   Si publica copias impresas (o copias en soportes que tengan normalmente cubiertas impresas) del Documento que sobrepasen las 100, y el aviso de licencia del Documento exige Textos de Cubierta, debe incluir las copias con cubiertas que lleven en forma clara y legible todos esos Textos de Cubierta: Textos de Cubierta Delantera en la cubierta delantera y Textos de Cubierta Trasera en la cubierta trasera. Ambas cubiertas deben identificarlo a Usted clara y legiblemente como editor de tales copias. La cubierta debe mostrar el título completo con todas las palabras igualmente prominentes y visibles. Además puede añadir otro material en las cubiertas. Las copias con cambios limitados a las cubiertas, siempre que conserven el título del Documento y satisfagan estas condiciones, pueden considerarse como copias literales en otros aspectos.

   Si los textos requeridos para la cubierta son muy voluminosos para que quepan legiblemente, debe colocar los primeros (tantos como sea razonable colocar) en la verdadera cubierta, y situar el resto en páginas adyacentes.

   Si usted publica o distribuye copias Opacas del Documento cuya cantidad exceda las 100, debe incluir una copia Transparente legible por una máquina con cada copia Opa-

ca, o bien mostrar en cada copia Opaca una dirección de red a la que el público general que use redes tenga acceso de descarga usando protocolos de red públicos y estandarizados a una copia Transparente del Documento completa, sin material adicional. Si usted hace uso de la última opción, deberá tomar las medidas necesarias, cuando comience la distribución de las copias Opacas en cantidad, para asegurar que esta copia Transparente permanecerá accesible en el sitio establecido por lo menos un año después de la última vez que distribuya una copia Opaca (directamente o a través de sus agentes o distribuidores) de esa edición al público.

Se solicita, aunque no es requisito, que se ponga en contacto con los autores del Documento antes de redistribuir gran número de copias, para darles la oportunidad de que le proporcionen una versión actualizada del Documento.

4. MODIFICACIONES

Puede copiar y distribuir una Versión Modificada del Documento bajo las condiciones de las secciones 2 y 3 anteriores, siempre que usted publique la Versión Modificada bajo esta misma Licencia, con la Versión Modificada haciendo el rol del Documento, por lo tanto dando licencia de distribución y modificación de la Versión Modificada a quienquiera posea una copia de la misma. Además, debe hacer lo siguiente en la Versión Modificada:

A. Usar en la Página de Título (y en las cubiertas, si hay alguna) un título distinto al del Documento y de sus versiones anteriores (que deberían, si hay alguna, estar listadas en la sección de Historia del Documento). Puede usar el mismo título de versiones anteriores al original siempre y cuando quien las publicó originalmente otorgue permiso.

B. Listar en la Página de Título, como autores, a una o más personas o entidades responsables de la autoría de las modificaciones de la Versión Modificada, junto con por lo menos cinco de los autores principales del Documento (todos sus autores principales, si tiene menos de cinco), a menos que le eximan de tal requisito.

C. Mostrar en la página de Título el nombre del editor de la Versión Modificada.

D. Conservar todas las notas de copyright del Documento.

E. Añadir una nota de copyright apropiada a sus modificaciones, adyacente a las otras notas de copyright.

F. Incluir, inmediatamente después de las notas de copyright, una nota de licencia dando el permiso para usar la Versión Modificada bajo los términos de esta Licencia, en la forma mostrada en el Adendo abajo.

G. Conservar en esa nota de licencia el listado completo de las Secciones Invariantes y de los Textos de Cubierta que sean requeridos en el aviso de Licencia del Documento.

H. Incluir una copia sin modificación de esta Licencia.

I. Conservar la sección Titulada «Historia», conservar su Título y añadirle un elemento que declare al menos el título, el año, los nuevos autores y el editor de la Versión Modificada, tal como figuran en la Página de Título. Si no hay una sección Titulada Historia en el Documento, crear una estableciendo el título, el año, los autores y el editor del Documento, tal como figuran en su Portada, añadiendo además un elemento describiendo la Versión Modificada, como se estableció en la oración anterior.

J. Conservar la dirección en red, si la hay, dada en el Documento para el acceso público a una copia Transparente del Documento, así como las otras direcciones de red dadas en el Documento para versiones anteriores en las que estuviese basado. Pueden ubicarse en la sección «Historia». Se puede omitir la ubicación en red de un trabajo que haya sido publicado por lo menos cuatro años antes que el Documento mismo, o si el editor original de dicha versión da permiso.

K. En cualquier sección Titulada «Agradecimientos» o «Dedicatorias», Conservar el Título de la sección y conservar en ella toda la sustancia y el tono de los agradecimientos y/o dedicatorias incluidas allí.

L. Conservar todas las Secciones Invariantes del Documento, sin alterar su texto ni sus títulos. Los números de sección o el equivalente no son considerados parte de los títulos de la sección.

M. Borrar cualquier sección titulada «Aprobaciones». Tales secciones no pueden incluirse en las Versiones Modificadas.

N. No cambiar el título de ninguna sección «Aprobaciones» autorizada ni entrar en conflicto con el título de alguna Sección Invariante.

O. Conservar todas las Limitaciones de Garantía.

Si la Versión Modificada incluye secciones de texto preliminar o apéndices nuevos que califiquen como Secciones Secundarias y contienen material no copiado del Documento, puede opcionalmente designar algunas o todas esas secciones como invariantes. Para hacerlo, añada sus títulos a la lista de Secciones Invariantes en la nota de licencia de la Versión Modificada. Tales títulos deben ser distintos de cualquier otro título de sección.

Puede añadir una sección titulada «Aprobaciones», siempre que contenga únicamente aprobaciones de su Versión Modificada por otras fuentes —por ejemplo, observaciones de peritos o que el texto ha sido aprobado por una organización como la definición oficial de un estándar—.

Puede añadir un pasaje de hasta cinco palabras como Texto de Cubierta Delantera y un pasaje de hasta 25 palabras como Texto de Cubierta Trasera al final de la lista de Textos de Cubierta en la Versión Modificada. Una entidad solo puede añadir (o hacer que se añada) un pasaje al Texto de Cubierta Delantera y uno al de Cubierta Trasera. Si el Documento ya incluye un textos de cubiertas añadidos previamente por usted o por la misma entidad que usted representa, usted no puede añadir otro; pero puede reemplazar el anterior, con permiso explícito del editor que agregó el texto anterior.

Con esta Licencia ni el/los autor(es) ni el/los editor(es) del Documento dan permiso para usar sus nombres para publicidad ni para asegurar o implicar aprobación de cualquier Versión Modificada.

5. COMBINACIÓN DE DOCUMENTOS

Usted puede combinar el Documento con otros documentos publicados bajo esta Licencia, bajo los términos definidos en la sección 4 anterior para versiones modificadas, siempre que incluya en la combinación todas las Secciones Invariantes de todos los documentos originales, sin modificar, listadas todas como Secciones Invariantes del trabajo combinado en su nota de licencia. Así mismo, debe incluir la Limitación de Garantía.

El trabajo combinado necesita contener solamente una copia de esta Licencia, y puede reemplazar varias Secciones Invariantes idénticas por una sola copia. Si hay varias

Secciones Invariantes con el mismo nombre pero con contenidos diferentes, haga el título de cada una de estas secciones único añadiéndole al final del mismo, entre paréntesis, el nombre del autor o editor original de esa sección, si es conocido, o si no, un número único. Haga el mismo ajuste a los títulos de sección en la lista de Secciones Invariantes de la nota de licencia del trabajo combinado.

En la combinación, debe combinar cualquier sección Titulada «Historia» de los documentos originales, formando una sección Titulada «Historia»; de la misma forma combine cualquier sección Titulada «Agradecimientos», y cualquier sección Titulada «Dedicatorias». Debe borrar todas las secciones tituladas «Aprobaciones».

6. COLECCIONES DE DOCUMENTOS

Puede hacer una colección que conste del Documento y de otros documentos publicados bajo esta Licencia, y reemplazar las copias individuales de esta Licencia en todos los documentos por una sola copia que esté incluida en la colección, siempre que siga las reglas de esta Licencia para cada copia literal de cada uno de los documentos en cualquiera de los demás aspectos.

Puede extraer un solo documento de una de tales colecciones y distribuirlo individualmente bajo esta Licencia, siempre que inserte una copia de esta Licencia en el documento extraído, y siga esta Licencia en todos los demás aspectos relativos a la copia literal de dicho documento.

7. AGREGACIÓN CON TRABAJOS INDEPENDIENTES

Una recopilación que conste del Documento o sus derivados y de otros documentos o trabajos separados e independientes, en cualquier soporte de almacenamiento o distribución, se denomina un «agregado» si el copyright resultante de la compilación no se usa para limitar los derechos legales de compilación de los usuarios más allá de lo que los de los trabajos individuales permiten. Cuando el Documento se incluye en un agregado, esta Licencia no se aplica a los otros trabajos del agregado que no sean en sí mismos derivados del Documento.

Si el requisito de la sección 3 sobre el Texto de Cubierta es aplicable a estas copias del Documento y el Documento es menor que la mitad del agregado entero, los Textos de Cubierta del Documento pueden colocarse en cubiertas que enmarquen solamente el Documento dentro del agregado, o el equivalente electrónico de las cubiertas si el documento está en forma electrónica. En caso contrario deben aparecer en cubiertas impresas enmarcando todo el agregado.

8. TRADUCCIÓN

La Traducción es considerada como un tipo de modificación, por lo que usted puede distribuir traducciones del Documento bajo los términos de la sección 4. El reemplazo de las Secciones Invariantes con traducciones requiere permiso especial de los dueños de su copyright, pero usted puede añadir traducciones de algunas o todas las Secciones Invariantes a las versiones originales de las mismas. Puede incluir una traducción de esta Licencia, de todas los avisos de licencia del documento, así como de las Limitaciones de Garantía, siempre que incluya también la versión en inglés de esta Licencia y las versiones originales de las notas de licencia y Limitaciones de Garantía. En caso de desacuerdo entre la traducción y la versión original de esta Licencia, la nota de licencia o la limitación de garantía, la versión original prevalecerá.

Si una sección del Documento se Titula «Agradecimientos», «Dedicatorias» o «Historia» el requisito (sección 4) de Conservar su Título (Sección 1) requerirá, típicamente, cambiar su título.

9. TERMINACIÓN

   Usted no puede copiar, modificar, sublicenciar o distribuir el Documento salvo por lo permitido expresamente bajo esta Licencia. Cualquier intento en otra manera de copia, modificación, sublicenciamiento o distribución de él es nulo, y dará por terminados automáticamente sus derechos bajo esa Licencia.

   Sin embargo, si usted cesa toda violación a esta Licencia, entonces su licencia proveniente de un titular de copyright queda restaurada (a) provisionalmente, a menos y hasta que el titular del copyright explicita y finalmente termine su licencia, y (b) permanentemente, si el titular del copyright falla en notificarle de la violación por algún medio razonable en un tiempo menor a 60 días después del cese.

   Además, su licencia proveniente de un titular del copyright particular queda restaurada permanentemente si el titular del copyright lo notifica de la violación por algún método razonable, es la primera vez que usted ha recibido aviso de la violación de esta Licencia (para cualquier trabajo) de ese titular del copyright, y usted remedia la violación en un tiempo menor a 30 días después de recibir dicho aviso.

   La terminación de sus derechos bajo esta sección no termina la licencia de terceros que hayan recibido copias o derechos de usted bajo ésta Licencia. Si sus derechos han sido terminados y no restaurados permanentemente, recibir una copia de alguna parte o el total del mismo material no le da ningún derecho de usarlo.

10. REVISIONES FUTURAS DE ESTA LICENCIA

    De vez en cuando la Free Software Foundation puede publicar versiones nuevas y revisadas de la GNU Free Documentation License. Tales versiones nuevas serán similares en espíritu a la presente versión, pero pueden diferir en detalles para solucionar nuevos problemas o intereses. Vea `https://www.gnu.org/licenses/`.

    A cada versión de la Licencia se le da un número de versión distintivo. Si el Documento especifica que se aplica una versión numerada en particular de esta licencia «o cualquier versión posterior», usted tiene la opción de seguir los términos y condiciones de la versión especificada o cualquier versión posterior que haya sido publicada (no como borrador) por la Free Software Foundation. Si el Documento no especifica un número de versión de esta Licencia, puede escoger cualquier versión que haya sido publicada (no como borrador) por la Free Software Foundation. Si el Documento especifica que un apoderado puede decidir qué versión futura de esta Licencia puede ser utilizada, esa frase de aceptación del apoderado de una versión le autoriza permanentemente a escoger esa versión para el Documento.

11. CAMBIO DE LICENCIA

    Un «Sitio de Colaboración Masiva Multiautor» (o «Sitio CMM») significa cualquier servidor World Wide Web que publique trabajos que puedan ser sujetos a copyright y que también provea medios prominentes para que cualquiera pueda editar esos trabajos. Una wiki pública que cualquiera puede editar es un ejemplo de tal servidor. Una «Colaboración Masiva Multiautor» (o «CMM») contenida en el sitio significa cualquier colección de trabajos que puedan ser sujetos a copyright publicados en el sitio de CMM.

«CC-BY-SA» significa la licencia Creative Commons Attribution-Share Alike 3.0 publicada por Creative Commons Corporation, una corporación sin fines de lucro con base en San Francisco, California, así como versiones futuras copyleft de esa licencia publicada por esa misma organización.

«Incorporar» significa publicar o republicar un Documento, como un todo o parcialmente, como parte de otro Documento.

Un sitio CMM es «elegible para re-licenciamiento» si es licenciado bajo esta Licencia, y si todos los trabajos que fueron publicados originalmente bajo esta Licencia en algún otro lugar diferente a esta CMM, y subsecuentemente incorporado como un todo o parcialmente a la CMM, (1) no tenía textos de cubierta o secciones invariantes, y (2) fueron incorporados antes del 1 de noviembre de 2008.

El operador de un Sitio CMM puede volver a publicar una CMM contenida en el sitio bajo CC-BY-SA en el mismo sitio en cualquier momento antes del 1 de agosto de 2009, siempre que la CMM sea elegible para relicenciamiento.

ADENDO: Cómo usar esta Licencia en sus documentos

Para usar esta Licencia es un documento que haya escrito, incluya una copia de la Licencia en el documento y ponga las siguientes notas de copyright y licencia justo después de la página de título:

```
Copyright (C)  año  su nombre.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

Si tiene Secciones Invariantes, Textos de Cubierta Delantera y Textos de Cubierta Trasera, reemplace la frase «with. . . Texts.» por esto:

```
with the Invariant Sections being lista de sus títulos, with
the Front-Cover Texts being lista, and with the Back-Cover Texts
being lista.
```

Si tiene Secciones Invariantes sin Textos de Cubierta o cualquier otra combinación de los tres, mezcle ambas alternativas para adaptarse a la situación.

Si su documento contiene ejemplos no triviales de código de programa, recomendamos publicar estos ejemplos en paralelo bajo su elección de licencia de software libre, como la GNU General Public License, para permitir su uso en software libre.

Apéndice E Glosarios

E.1 Índice de Instrucciones Integradas del Intérprete

S

T

U

W

E.2 Índice de Palabras Reservadas del Intérprete

!

[

]

{

}

C

D

E

F

I

S

T

U

W

E.3 Índice de Parámetros y Variables

!

#

$

*

—

?

@

–

0

A

B

C

D

E

F

G

H

I

K

L

M

O

P

R

S

T

U

V

E.4 Índice de Funciones

A

B

C

D

E

F

G

H

I

K

M

N

O

P

Q

R

S

T

U

Y

E.5 Índice Conceptual

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V