Bash Prompt COMO

Giles Orr, *giles@interlog.com*
Traducido por Iosu Santurtún, *iosu@bigfoot.com* v0.60, 7 de Enero de 1999, Traducción 12 de Junio de 1999

En este documento se comenta la creación y el manejo de prompts de terminales en modo texto y X, incluyendo secuencias estándar de escape que proporcionan el nombre de usuario, el directorio actual de trabajo, la hora, etc. Se hacen sugerencias más complejas sobre cómo modificar las barras de título de las terminales X, cómo usar funciones externas para proporcionar información en el prompt, y cómo usar colores ANSI.

Índice General

1 Introducción y «administrivia»

1.1 Requerimientos

Será necesario el `bash`. La versión por defecto de la práctica totalidad de distribuciones LiNUX es la `1.14.7`, que es una versión bien conocida y de confianza. Actualmente se encuentra disponible la versión `2.0` (incluso superiores): yo llevo usando la `2.0` algún tiempo, pero la mayoría del código aquí presentado debería funcionar bajo la `1.14.7`. Si conozco alguna incompatibilidad, lo mencionaré. Se puede comprobar la versión del `bash` mediante el comando `echo $BASH_VERSION`. En mi máquina responde `2.02.1(1)-release`.

La experiencia en programación `shell` puede venir bien, pero no es esencial: cuanto más se sepa, más complejos serán los prompts que se puedan crear. Presupongo un conocimiento básico de la programación `shell` y utilidades unix a lo largo de este tutorial. Sin embargo, mis propios niveles de programación `shell` son limitados, así que doy gran cantidad de ejemplos y explicaciones que pueden parecer innecesarias para el programador experimentado.

1.2 Cómo usar este documento

Se incluyen muchos ejemplos y textos aclaratorios. Su utilidad variará según la persona de la que se trate. Esto ha crecido lo suficiente como para que una lectura completa pueda resultar difícil; se recomienda leer únicamente las secciones que se necesite, volviendo hacia atrás las veces que sea necesario.

1.3 Traducciones

A fecha de 6 de Enero de 1999, existen traducciones al japonés (Akira Endo, *akendo@t3.rim.or.jp*) y alemán (Thomas Keil, *thomas@h-preissler.de*). ¡Gracias a ambos! Las URL serán incluidas cuando estén disponibles.[1]

1.4 Problemas

Esta es una lista de problemas que he encontrado programando *prompts*. No comience a leer por aquí, y no deje que esta lista le desanime, la mayoría son detalles de poca importancia. Échele un vistazo únicamente si llega a algún punto conflictivo.

- Algunas características de `bash` (tales como funciones matemáticas dentro de `$(())`, entre otras) son opciones en tiempo de compilación. Si está usando una distribución binaria, como la que viene en las distribuciones estándar de LiNUX, tales características deberían estar incluidas. Pero si está trabajando en otro sistema, merece la pena recordar esto si no funciona algo que debería hacerlo. Algunas notas acerca de esto en *Learning the Bash Shell*, págs *260-262*
- El manejador de terminal `screen` no siempre funciona con colores ANSI. Desafortunadamente no soy un experto en `screen`. Mi versión de `screen` (una muy reciente) parece que funciona bien en estos casos, pero he visto ocasiones en que redujo todos los colores del prompt al color de primer plano estándar en terminales X. Esto no parece ser un problema en la consola.
- Los ficheros `Xdefaults` pueden redefinir colores. Mire en `~/.Xdefaults` las líneas referidas a `XTerm*background` y `XTerm*foreground` (o posiblemente `XTerm*Background` y `XTerm*Foreground`).
- Uno de los *prompts* que se comentan en este documento utiliza la salida de `jobs` - como se comenta en su momento, las salida de `jobs` a una tubería no funciona con `bash 2.02`.
- Las secuencias de escape ANSI de movimiento del cursor no están implementadas en todas las terminales X. Esto se comenta en su propia sección.
- Se pueden crear «pseudo-gráficos» bastante agradables utilizando una fuente VGA en lugar de las estándar de LiNUX. Desafortunadamente, estos efectos son horrorosos si no se utiliza un tipo VGA, y no hay manera de detectar dentro de una terminal qué clase de fuentes de letra se están utilizando.
- Ha aparecido `bash 2.02+` que incorpora nuevas características, y cambia algunos comportamientos. Lo que funcionase bajo la versión `1.14.7` no tiene por qué funcionar bajo la `2.0+`, y viceversa.

1.5 Comentarios y sugerencias

Esta es una experiencia de aprendizaje para mi. He llegado a saber bastante acerca de lo que se puede hacer para crear *prompts* interesantes y útiles, pero necesito indicaciones para corregir y mejorar este documento. He intentado comprobar las sugerencias que yo hago contra diferentes versiones de `bash` (sobre todo la `2.02` que uso, y la `1.14.7` que es la más popular), no obstante, comuníqueme cualquier incompatibilidad que encuentre.

La última versión de éste documento debería estar siempre disponible en
http://www.interlog.com/~giles/bashprompt.html. Compruébelo y siéntase libre de mandarme un mensaje a *giles@interlog.com* con sus sugerencias.

[1]N. del T.
bueno, ahora también está disponible la versión en castellano ;)

Utilizo los HOWTOs del *Linux Documentation Project* casi exclusivamente en formato HTML, así que cuando los convierto desde SGML, el HTML es el único formato que compruebo concienzudamente. Si hay problemas con otros formatos puede que yo no sepa nada, por lo que agradecería cualquier comentario sobre ello.

1.6 Créditos

En la creación de este documento he tomado prestado mucho del proyecto *BashPrompt*, disponible en *http://bash.current.nu*.

Otras fuentes han sido:

- mini-COMO acerca del *Título de las* `xterm` de Ric Lister, disponible en *http://sunsite.unc.edu/LDP/HOWTO/mini/Xterm-Title.html*.
- *Prompts ANSI* de Keebler, disponible en *http://www.ncal.verio.com/~keebler/ansi.html*.
- *Cómo hacer un prompt para* `bash` de Stephen Webb, disponible en *http://bash.current.nu/bash/HOWTO.html*.
- y *X ANSI Fonts* de Stumpy, disponible en *http://home.earthlink.net/~us5zahns/enl/ansifont.html*

También han sido de inmensa ayuda numerosas conversaciones y e-mails de Dan, un compañero del *Georgia College & State University*, cuyo conocimiento sobre unix sobrepasa el mío con mucho. Me ha proporcionado excelentes sugerencias; ideas suyas han conducido hacia prompts interesantes.

Tres libros que me han has sido de mucha utilidad programando prompts son:

- *Linux in a Nutshell* de Jessica Heckman Perry (*O'Reilly*, 1997)
- *Learning the Bash Shell* de Cameron Newham y Bill Rosenblatt (*O'Reilly*, 2a. ed., 1998)
- y *Unix Shell Programming* de Lowell Jay Arthur (*Wiley*, 1986; esta es la primera edición, la segunda apareció en 1997).

1.7 Copyright y demás

Este documento es copyright ©1998-1999 de Giles Orr. Se anima a su distribución, aunque no debería modificarse este documento (véase la sección 1.5 (Comentarios y Sugerencias) para todo lo referente a ponerse en contacto conmigo: he venido añadiendo los cambios sugeridos por los lectores desde hace mucho). Póngase en contacto conmigo si está interesado en realizar una traducción, esa es una de las modificaciones con las que puedo vivir.

2 bash y sus prompts

2.1 ¿Qué es bash?

Descendiente del *Bourne Shell*, `bash` es un producto GNU, el *Bourne Again Shell*. Es el interfaz estándar de línea de comandos en la mayoría de las máquinas LiNUX. Potencia la interactividad, soportando edición en línea de comando, capacidad de completar o recordar automáticamente un comando, etc. También soporta prompts configurables - la mayoría de la gente se da cuenta de esto, pero no saben hasta qué punto.

2.2 ¿Qué puede aportar la manipulación del prompt ?

La mayoría de los sistemas LiNUX tienen un prompt por defecto en un solo color (normalmente gris) que indica el nombre de usuario, el nombre de la máquina en la que se está trabajando y alguna indicación acerca del directorio de trabajo actual. Toda esta información es útil, pero se puede ir mucho más allá: se puede mostrar todo tipo de información (número de «tty», hora, fecha, carga, número de usuarios, tiempo sin reiniciar ...) y el prompt puede usar colores ANSI, ya sea por razones puramente estéticas, o para remarcar cierta información. También se puede manipular la barra de título de una `xterm` para reflejar parte de esta información.

2.3 ¿Por qué molestarse ?

Además de una apariencia bonita, en ocasiones es útil seguir la pista de cierta información del sistema. Una idea que sé que gusta a la gente es que es posible poner los prompts de diferentes máquinas en diferentes colores. Si se tienen varias terminales X abiertas en diferentes máquinas, o si se tiende a olvidar en qué máquina se está trabajando y se borran ficheros equivocados, encontrará en esta una buena forma de recordar en qué máquina se encuentra.

2.4 El primer paso

La apariencia del prompt viene dada por la variable del *shell* `PS1`. Las continuaciones de comandos se indican mediante la cadena `PS2`, que puede modificarse de la misma forma que aquí se comentan –ya que el manejo es exactamente el mismo, y que no es tan «interesante»–, casi siempre se van a tratar modificaciones de la cadena `PS1` (También existen las cadenas `PS3` y `PS4`. Estas nunca están a la vista del usuario medio (vea la página del manual sobre `bash` si está interesado en su propósito).

Para cambiar el aspecto del prompt, hay que cambiar la variable `PS1`. Para experimentar, se pueden introducir cadenas `PS1` directamente desde el prompt, y ver los resultados inmediatamente (esto sólo afecta a la sesión actual, y los cambios desaparecen cuando termina). Si se desea hacer permanentes estos cambios, modifique su `~/.bashrc`, y añada la nueva definición de `PS1`. Si tiene permisos de `root`, puede mirar en `/etc/profile` y modificar allí la línea `PS1=`. Tenga en cuenta que en algunas distribuciones (al menos en la *RedHat 5.1*) `/etc/bashrc` redefine los valores de `PS1` y `PS2`.

Antes de comenzar, es importante recordar que la cadena `PS1` se almacena en el entorno como cualquier otra variable de entorno. Si se modifica en la línea de comando, su prompt cambiará. Antes de hacer cualquier cambio, puede salvar su prompt actual en otra variable de entorno.

```
[giles@nikola giles]$ SAVE=$PS1
[giles@nikola giles]$
```

El prompt más sencillo sería el de un sólo carácter, como:

```
[giles@nikola giles]$ PS1=$
$ls
bin   mail
$
```

Esto demuestra la mejor manera de experimentar con prompts básicos: introduciéndolos en la línea de comando. Nótese que el texto introducido por el usuario aparece inmeditamente después del prompt. Yo prefiero usar

```
$PS1="$ "
$ ls
bin   mail
$
```

que fuerza un espacio después del prompt, haciéndolo más legible. Para restaurar el prompt original, basta con llamar a la variable almacenada:

```
$ PS1=$SAVE
[giles@nikola giles]$
```

2.5 Secuencias de escape del prompt `bash`

Hay numerosas secuencias de escape ofrecidas por el shell `bash` para insertar en el prompt. De la página del manual del `bash 2.02`:

Cuando se ejecuta interactivamente, `bash` muestra el prompt primario `PS1` cuando está listo para leer un comando, y el prompt secundario `PS2` cuando necesita más datos de entrada para completar un comando. `bash` permite que estas cadenas de prompt sean modificadas insertando ciertos caracteres especiales escapados mediante contrabarra que se decodifican de la manera siguiente:

- `\a` carácter de campana ASCII (`07`)
- `\d` la fecha en formato *día mes día* (p.ej., `mar may 26`)
- `\e` caracter de escape ASCII (`033`)
- `\h` el nombre del host hasta el primer «.»
- `\H` el nombre del la máquina completo (*FQDN*)
- `\n` caracter de nueva línea
- `\r` retorno de carro
- `\s` el nombre del shell, el nombre base de `$0` (el fragmento que sigue a la última barra)
- `\t` la hora actual en formato 24-horas `HH:MM:SS`
- `\T` la hora actual en formato 12-horas `HH:MM:SS`
- `\@` la hora actual en formato 12-horas AM/PM
- `\u` el nombre de usuario del usuario actual
- `\v` la versión de `bash` (p.ej., `2.0`)
- `\V` la versión del paquete del `bash`, versión + *patch-level* (p.ej., `2.00.0`)
- `\w` el directorio actual de trabajo
- `\W` el nombre base del directorio actual de trabajo
- `\!` el número del comando actual en el histórico
- `#` el número de comando del comando actual
- `\$` si el UID efectivo es `0`, un `#`; en otro caso, `$`

- \nnn el caracter correspondiente al número en octal nnn
- \\ una contrabarra
- \[inicio de una secuencia de caracteres no imprimibles que pueden usarse para incrustar una secuencia de control del terminal en el prompt.
- \] fin de una secuencia de caracteres no imprimibles

Continuando donde lo habíamos dejado:

```
[giles@nikola giles]$ PS1="\u@\h \W> "
giles@nikola giles> ls
bin   mail
giles@nikola giles>
```

Este es similar al prompt por defecto de la mayoría de las distribuciones LiNUX. Pero yo quería una apariencia ligeramente diferente, así que lo cambié a:

```
giles@nikola giles> PS1="[\t][\u@\h:\w]\$ "
[21:52:01][giles@nikola:~]$ ls
bin   mail
[21:52:15][giles@nikola:~]$
```

2.6 Valor permanente de las cadenas «PS?»

Las cadenas `PS?` son establecidas, según la persona o distribución en distintos lugares. Los más comunes son `/etc/profile`, `/etc/bashrc`, `~/.bash_profile`, y `~/.bashrc`. Johan Kullstam, ***johan19@idt.net*** escribe:

La cadena `PS1` *debería ponerse en el* `.bashrc` *debido a que los* `bash` *no interactivos no tienen en cuenta este fichero, y por tanto, no estableceremos* `PS1` *si se trata de una shell no interactiva. La página del manual de* `bash` *indica que la presencia o ausencia de* `PS1` *es una buena manera de saber si uno está en una sesión interactiva o no interactiva de bash.*

La forma en que me percaté de esto es que `startx` *es un script bash, lo que implica que pulverizará el prompt. Cuando se pone* `PS1` *en el* `.profile` *(o en el* `.bash_profile`*, al entrar en consola y lanzar las X vía* `startx`*, la variable* `PS1` *se elimina en el proceso, dejándole con el prompt por defecto.*

Una solución es lanzar las `xterm` *y las* `rxvt` *con la opción* `-ls` *para forzarles a leer el* `.profile`*, pero en el momento en que se llame un shell mediante un shell-script no interactivo se perderá* `PS1`. `system(3)` *usa* `sh -c` *que, si* `sh` *es* `bash`*, eliminará* `PS1`*. Una forma mejor de hacer esto es situar la definición de* `PS1` *en* `.bashrc`*. Este fichero se lee cada vez que se inicia bash y es donde deberían aparecer las cosas interactivas (p.ej.* `PS1`*)*

Por lo tanto se llega a la conclusión de que `PS1= ...(blah)...` *debería ir en* `.bashrc` *y no en* `.profile`

He intentado simular el problema que él comenta, y he encontrado uno diferente: mi variable `PROMPT_COMMAND` (de la que se hablará después) era desintegrada. Mi conocimiento en este área es un poco limitado, así que seguiré lo que dice Johan.

3 Comandos Externos

3.1 PROMPT_COMMAND

`bash` proporciona otra variable de entorno llamada `PROMPT_COMMAND`. El contenido de esta variable se ejecuta como un comando **bash** normal justo antes de que **bash** muestre el prompt.

```
[21:55:01][giles@nikola:~] PS1="[\u@\h:\w]\$ "
[giles@nikola:~] PROMPT_COMMAND="date +%H%M"
2155
[giles@nikola:~] d
bin   mail
2156
[giles@nikola:~]
```

Lo que ocurre arriba es que he cambiado `PS1` para que no incluya la secuencia de escape `\t`, de tal modo que la hora no forme parte del prompt. Después he usado `date +%H%M` para mostrar la hora en un formato que me gusta más. Pero aparece en una línea diferente a la del prompt. Esto se soluciona usando `echo -n ...` como se muestra debajo, funciona con `bash 2.0+`, pero parece que no lo hace con `bash 1.14.7`: aparentemente el prompt se dibuja de manera diferente, y el método mostrado a continuación resulta en superposición de texto.

```
2156
[giles@nikola:~] PROMPT_COMMAND="echo -n [$(date +%H%M)]"
[2156][giles@nikola:~]$
[2156][giles@nikola:~]$ d
bin   mail
[2157][giles@nikola:~]$ unset PROMPT_COMMAND
[giles@nikola:~]
```

`echo -n ...` controla la salida del comando `date` y suprime el caracter de nueva línea final, permitiendo que el prompt aparezca en una sola línea. Al final, uso el comando `unset` para eliminar la variable de entorno `PROMPT_COMMAND`.

Nótese que uso la convención $(<comando>) para la sustitución de comandos, es decir

```
$(date +%H%M)
```

significa «sustituye la salida de `date +%H%M` aquí». Esto funciona en `bash 2.0+`. En alguna versión antigua de `bash`, anterior a la `1.14.7`, puede ser necesario el uso de comillas simples graves (`` `date +%H%M` ``). Estas comillas pueden usarse en `bash 2.0+`, pero es preferible usar `$()`, que funciona mejor en el caso de anidamientos. Voy a usar esta convención a lo largo de este documento. Si utiliza una versión anterior de `bash`, normalmente podrá sustituir los `$()` por las comillas. Si la sustitución de comandos está escapada (es decir, `\$(comando)`), entonces deberá usar contrabarras para escapar AMBAS comillas (o sea, `` \`comando\` ``).

3.2 Comandos externos en el prompt

También se puede usar la salida de comandos regulares LiNUX directamente en el prompt. Obviamente, no es deseable insertar muchas cosas, o se creará un prompt enorme. Además será preferible usar un comando rápido ya que se va a ejecutar cada vez que el prompt aparezca en pantalla, y retrasa la aparición de éste lo que puede resultar muy molesto. (A diferencia del ejemplo anterior al que recuerda, esto funciona con `bash 1.14.7`)

```
[21:58:33][giles@nikola:~]$ PS1="[\$(date +%H%M)][\u@\h:\w]\$ "
[2159][giles@nikola:~]$ ls
bin   mail
[2200][giles@nikola:~]$
```

Es importante hacer notar la contrabarra anterior al signo de dólar de la sustitución del comando. Sin ella, el comando externo se ejecuta exactamente una vez: cuando se lee la cadena almacenada en `PS1` del entorno. Para este prompt, eso significaría que mostraría siempre la misma hora, sin importar cuanto tiempo se ha usado el prompt. La contrabarra protege los contenidos de `$()` de la interpretación inmediata del shell, por lo que `date` es llamado cada vez que se genera un prompt.

LiNUX incluye muchas utilidades de pequeño tamaño como `date`, `grep` o `wc` que permiten la manipulación de datos. Si se encuentra en la situación de crear una combinación compleja de estos programas dentro del prompt, podría ser más fácil crear un *shell script* y llamarlo desde el prompt. En ocasiones son necesarias secuencias de escape en los scripts de `bash` para asegurar que las variables se expanden en el momento correcto (como se ha mostrado arriba con el comando `date`): esto llega a niveles mayores con la línea de prompt `PS1`, y es una buena idea evitarlo creando *scripts*.

Un ejemplo de un pequeño *shell script* usado dentro de un prompt es el siguiente:

```
#!/bin/bash
#     lsbytesum - suma del número total de bytes de un ls
TotalBytes=0
for Bytes in $(ls -l | grep "^-" | cut -c30-41)
do
    let TotalBytes=$TotalBytes+$Bytes
done
TotalMeg=$(echo -e "scale=3 \n$TotalBytes/1048576 \nquit" | bc)
echo -n "$TotalMeg"
```

A veces he mantenido ambos como funciones (mucho más eficiente, pero desafortunadamente, la explicación de funciones en detalle va más allá de este documento), o como *scripts* en mi directorio `/bin`, que se encuentra en mi variable `PATH`. Utilizándolo en un prompt:

```
[2158][giles@nikola:~]$ PS1="[\u@\h:\w (\$(lsbytesum) Mb)]\$ "
[giles@nikola:~ (0 Mb)]$ cd /bin
[giles@nikola:/bin (4.498 Mb)]$
```

3.3 Qué poner en el prompt

Se habrá percatado de que yo pongo el nombre de usuario, el nombre de la máquina, la hora y el directorio actual en la mayoría de mis prompts. Con la excepción de la hora, son cosas muy normales de encontrar en un prompt, y la hora es posiblemente la adición más común. Pero lo que incluya cada uno es cosa de gusto personal. Aquí hay ejemplos de personas que conozco que le pueden dar ideas.

El prompt de Dan es mínimo pero muy efectivo, particularmente para su forma de trabajar.

```
[giles@nikola:~]$ cur_tty=$(tty | sed -e "s/.*tty\(.*\)/\1/")
[giles@nikola:~]$ echo $cur_tty
p4
[giles@nikola:~]$ PS1="\!,$cur_tty,\$?\$ "
1095,p4,0$
```

A Dan no le gusta que el hecho de tener el directorio actual de trabajo en el prompt pueda variar el tamaño de éste drásticamente mientras se pasa de un directorio a otro, así que el mantiene la pista de esto en su cabeza (o usa `pwd`). El aprendió Unix con `csh` y `tcsh`, así que usa su histórico de comandos de forma intensiva (cosa que los adictos al `bash` no solemos hacer), así que la primera cosa en el prompt es el número del histórico. El segundo campo es el caracter significante de la tty (la salida de `tty` es recortada mediante sed), un dato que puede ser útil para los usuarios de `screen`. El tercer campo es el valor de retorno del último comando/tubería (nótese que se muestra inútil para cualquier comando que se ejecuta dentro del prompt; se puede solucionar capturándolo en una variable). Finalmente, «`\$`» es un símbolo de dólar para un usuario normal y cambia a `#` si el usuario es el `root`.

Torben Fjerdingstad me escribió para decirme que a menudo suspende tareas, y después se le olvidan, así que usa su prompt para servir de recordatorio de las tareas suspendidas:

```
[giles@nikola:~]$ function jobcount {
> jobs|wc -l| awk '{print $1}'
> }
[giles@nikola:~]$ export PS1='\W[`jobcount`]# '
giles[0]# man ls &
[1] 4150

[1]+  Stopped (tty output)    man ls
giles[1]#
```

Torben usa `awk` para evitar el espacio de la salida de `wc`, mientras que yo habría usado `sed` o `tr` - no porque sean mejor, sino porque me resultan más familiares. Probablemente existan más formas. Torben además rodea sus cadenas `PS1` con comillas simples. lo que evita que el `bash` interprete inmediatamente las contrabarras, así no tiene que escaparlas como yo había dicho.

NOTA: existe un *bug* conocido en `bash 2.02` que provoca que el comando `jobs` no retorne nada a una tubería. Si intenta lo de arriba bajo `bash 2.02`, siempre obtendrá un «0» independientemente de los trabajos que haya suspendidos. Chet Ramey, uno de los responsables de `bash` me ha dicho que esto se soluciona en la `v2.03`.

3.4 Entorno `bash` y funciones

Como he mencionado antes, `PS1`, `PS2`, `PS3`, `PS4` y `PROMPT_COMMAND` se almacenan todas en el entorno del `bash`. Para aquellos que provengan del DOS, la idea de almacenar gran cantidad de código en el entorno es aterradora, ya que el entorno del DOS era pequeño, y no creció bien exactamente. Posiblemente haya límites prácticos en lo que se puede y debe poner en el entorno, pero no los conozco, y probablemente se está hablando de un par de órdenes de magnitud mayores de a lo que están acostumbrados los usuarios de DOS. Como dijo Dan:

En mi shell interactivo tengo 62 alias y 25 funciones. Mi regla es que si necesito algo únicamente para uso interactivo y puedo escribirlo bien en bash, hago de ello una función de shell (teniendo en cuenta que no pueda expresarse de manera sencilla como un alias). Si la gente se preocupa por la memoria no deberían estar usando `bash`*.* `bash` *es uno de los programas más grandes que ejecuto en mi máquina LiNUX (aparte de Oracle). Ejecute* `top` *algún tiempo y pulse «M» para ordenar por memoria, y compruebe lo cerca que está bash de la cima de la lista. O sea, que es ¡mayor que sendmail!... Recomienda que utilicen mejor* `ash`*o algo así.*

Supongo que estaba usando la consola el día que probó eso: ejecutando X y aplicaciones X obtendrá muchas cosas mayores que bash. Pero la idea es la misma: el entorno es algo para ser usado, sin preocupación de desbordarlo.

Me arriesgo a la censura de los gurús de unix cuando digo esto (por el delito de supersimplificación), pero las funciones son básicamente pequeños `shell scripts` que se cargan en el entorno con el propósito de una mayor eficiencia. Citando a Dan de nuevo: *las funciones shell son lo más eficiente. El procedimiento es similar a un* `source` *de un shell script pero con el ahorro de las operaciones entrada/salida, ya que la función* *se encuentra ya en memoria. Las funciones shell se cargan típicamente del* `.bashrc` o `.bash_profile` *dependiendo de si se las quiere en el shell inicial o en los sucesivos subshells también.*

Compárese esto con la ejecución de un shell script: el shell realiza un `fork`*, el hijo lleva a cabo un* `exec`*, potencialmente se busca el* `path`*, el kernel abre el fichero y examina la cantidad suficiente de bytes para saber cómo ejecutarlo, en el caso de un shell script debe arrancarse un shell con el nombre del script como argumento. Comparado con una función shell, cualquier cosa aparte de la ejecución de las sentencias, puede considerarse una sobrecarga innecesaria.*

4 Manipulaciones de la barra de título de Xterm

Pueden usarse secuencias de escape no imprimibles para producir efectos interesantes en los prompts. Para usar estas secuencias es necesario encerrarlas entre `\[` y `\]`, advirtiendo al `bash` de que ignore estas secuencias cuando calcule la longitud del prompt. No incluir estos delimitadores resulta en una colocación errónea del cursor en el código de edición de línea, ya que no conoce el tamaño real del prompt. Las secuencias de escape además deben ir precedidas de `\033[` en `bash` anteriores a la versión 2, o por `\033[` o `\[e` en versiones posteriores.

Si se intenta cambiar la barra de título de la Xterm con el prompt mientras se está en la consola, se producirá basura en el prompt. Para evitar esto, hay que comprobar la variable de entorno `TERM` para indicar que el prompt va a estar en una `xterm`:

```
function proml
{
case $TERM in
    xterm*)
        local TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        local TITLEBAR=''
        ;;
esac

PS1="${TITLEBAR}\
[\$(date +%H%M)]\
[\u@\h:\w]\
\$ "
PS2='> '
PS4='+ '
}
```

Esta es una función que puede ser añadida al `~/.bashrc`. Entonces se podrá invocar a la función mediante su nombre. La función, como la cadena `PS1`, se almacena en el entorno. Una vez que la función ha dado valor a la cadena `PS1`, se puede eliminar la función del entorno con `unset proml`. Debido a que el prompt no puede cambiar de estar en una Xterm a estar en la consola, la variable `TERM` no se comprueba cada vez que se genera el prompt. He usado marcadores de comtinuación (contrabarras, \) en la definición del prompt, para permitir continuarlo a lo largo de varias líneas. Esto mejora la legibilidad, haciendo más fácil modificarlo o depurarlo.

Lo he definido como una función porque es así como funciona el paquete `Bashprompt`: no es la única manera de hacerlo pero funciona bien. A medida que los prompts usados se tornen más complejos, resulta más y más tedioso teclearlos en la línea de comando, y más práctico situarlos en algún tipo de fichero de texto.

En este caso, para probar esto como prompt, salve lo de arriba como un fichero de texto llamado `proml`. Se puede trabajar con él de la manera que sigue:

```
[giles@nikola:/bin (4.498 Mb)]$ cd      -> Ir a donde se almacena el prompt
[giles@nikola:~ (0 Mb)]$ vi proml       -> Editar el prompt
      ...                               -> Introducir el texto anterior
[giles@nikola:~ (0 Mb)]$ source proml  -> Leer la funcion de prompt
[giles@nikola:~ (0 Mb)]$ proml         -> Ejecutar la funcion de
prompt
```

El primer paso en la creación del prompt es comprobar si el shell en el que nos encontramos es o no una `xterm`; si lo es, estará definida la variable de entorno `($(TITLEBAR))`. Esta consiste de la secuencia de escape apropiada, y `\u@;\h:\w`, lo que escribe `<usuario>@<máquina>:<directorio>` en la barra de título de la Xterm. Esto es particularmente útil para `xterm` minimizadas haciéndolas más rápidamente identificables. El resto de material de este prompt debería ser familiar de prompts previos que hemos creado.

El único percance que puede darse manipulando la barra de título de la `xterm` de esta forma ocurre cuando se ingresa en un sistema en el que no se ha preparado la barra de título: la `xterm` continuará mostrando la información del sistema anterior que tenía la barra de título manipulada.

5 Secuencias de escape ANSI: colores y movimientos del cursor

5.1 Colores

Como se mencionó antes, las secuencias escape de caracteres no imprimibles tienen que encerrarse entre `\[\033[` y `\]`. Para las secuencias de escape de color, también deben aparecer, seguidos además de una `m` minúscula.

Si se prueba uno de los prompts siguientes en una `xterm` y resulta que no se ven los colores nombrados, compruebe su `~/.Xdefaults` (y posiblemente sus hermanos) y busque líneas como `XTerm*Foreground: BlanchedAlmond`. Esto puede comentarse colocando un `!` delante. Por supuesto, depende de qué emulador de terminal esté usando. Este es el lugar más probable en el que los colores de su terminal pueden ser redefinidos.

Para incluir texto azul en el prompt

```
PS1="\[\033[34m\][\$(date +%H%M)][\u@\h:\w]$ "
```

El problema con este prompt es que el color azul que comienza con el código 34 no se retorna nunca al color habitual, por lo que cualquier texto que se teclee después del prompt será del mismo color que el prompt. Este es también un azul oscuro, así que combinarlo con el código de negrita puede resultar útil:

```
PS1="\[\033[1;34m\][\$(date +%H%M)][\u@\h:\w]$\[\033[0m\] "
```

Ahora el prompt es azul claro, y termina cambiando el color de nuevo a «nada» (el color que se tenía previamente de primer plano)

Aquí está el resto de equivalencias de colores:

```
Negro     0;30    Gris oscuro    1;30
Azul      0;34    Azul claro     1;34
Verde     0;32    Verde claro    1;32
```

```
Cyan        0;36    Cyan claro     1;36
Rojo        0;31    Rojo claro     1;31
Purpura     0;35    Purpura claro  1;35
Marron      0;33    Amarillo       1;33
Gris claro  0;37    blanco         1;37
```

También se pueden poner colores de fondo, usando 44 para fondo azul, 41 para fondo rojo, etc. No hay colores de fondo 'negrita'; se pueden usar combinaciones, como texto rojo claro sobre fondo azul `\[\033[44;1;31m\]`, aunque parece que funciona mejor poner los colores separadamente (es decir, `\[\033[44m\]\[\033[1;31m\]`). Otros códigos disponibles incluyen 4: subrayado, 5: parpadeante, 7: inverso y 8: oculto.

Nota: mucha gente (yo incluido), tienen fuertes objeciones al uso del atributo «parpadeo». Afortunadamente no funciona en ningún emulador de terminal que yo conozca - pero si que funciona en la consola. Y si alguien se preguntaba (como yo hice) «¿para qué sirve el atributo oculto?», yo he visto usarlo en un ejemplo de shell script (no en un prompt) que permitía introducir un password sin ser reflejado en la pantalla.

Basado en el prompt *elite2* del paquete `bashprompt` (que he modificado para funcionar mejor en una consola estándar, en lugar de con los tipos especiales de `xterm` necesarios para ver correctamente el original), este es un prompt que he usado mucho:

```
function elite
{

local GRAY="\[\033[1;30m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
local CYAN="\[\033[0;36m\]"
local LIGHT_CYAN="\[\033[1;36m\]"

case $TERM in
    xterm*)
        local TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        local TITLEBAR=""
        ;;
esac

local GRAD1=$(tty|cut -d/ -f3)
PS1="$TITLEBAR\
$GRAY-$CYAN-$LIGHT_CYAN(\
$CYAN\u$GRAY@$CYAN\h\
$LIGHT_CYAN)$CYAN-$LIGHT_CYAN(\
$CYAN\#$GRAY/$CYAN$GRAD1\
$LIGHT_CYAN)$CYAN-$LIGHT_CYAN(\
$CYAN\$(date +%H%M)$GRAY/$CYAN\$(date +%d-%b-%y)\
$LIGHT_CYAN)$CYAN-$GRAY-\
$LIGHT_GRAY\n\
$GRAY-$CYAN-$LIGHT_CYAN(\
$CYAN\$$GRAY:$CYAN\w\
$LIGHT_CYAN)$CYAN-$GRAY-$LIGHT_GRAY "
PS2="$LIGHT_CYAN-$CYAN-$GRAY-$LIGHT_GRAY "
}
```

Defino los colores como variables temporales del shell en favor de la legibilidad. Es más fácil trabajar así. La variable GRAD1 es una comprobación para determinar en qué terminal se está. Como la prueba para saber si se está en una xterm, solo es necesario llevarla a cabo una vez. El prompt es similar a esto, excepto el color

```
--(giles@nikola)-(75/ttyp7)-(1908/12-Oct-98)--
--($:~/tmp)--
```

Para recordar qué colores hay disponibles, uso el siguiente script que saca todos los colores por pantalla:

```
#!/bin/bash
#
# Este fichero saca por pantalla un monton de codigos de color
# para demostrar que hay disponible. Cada linea es un color con
# fondo negro y gris, con el codigo en medio. Funciona sobre
# fondos blancos, negros y verdes (2 dic. 98)
#
echo "  Sobre gris claro:        Sobre negro:"
echo -e "\033[47m\033[1;37m  Blanco        \033[0m\
 1;37m \
\033[40m\033[1;37m  Blanco        \033[0m"
echo -e "\033[47m\033[37m  Gris Claro    \033[0m\
   37m \
\033[40m\033[37m  Gris Claro    \033[0m"
echo -e "\033[47m\033[1;30m  Gris          \033[0m\
 1;30m \
\033[40m\033[1;30m  Gris          \033[0m"
echo -e "\033[47m\033[30m  Negro         \033[0m\
   30m \
\033[40m\033[30m  Negro         \033[0m"
echo -e "\033[47m\033[31m  Rojo          \033[0m\
   31m \
\033[40m\033[31m  Rojo          \033[0m"
echo -e "\033[47m\033[1;31m  Rojo Claro    \033[0m\
 1;31m \
\033[40m\033[1;31m  Rojo Claro    \033[0m"
echo -e "\033[47m\033[32m  Verde         \033[0m\
   32m \
\033[40m\033[32m  Verde         \033[0m"
echo -e "\033[47m\033[1;32m  Verde Claro   \033[0m\
 1;32m \
\033[40m\033[1;32m  Verde Claro   \033[0m"
echo -e "\033[47m\033[33m  Marrón        \033[0m\
   33m \
\033[40m\033[33m  Marron        \033[0m"
echo -e "\033[47m\033[1;33m  Amarillo      \033[0m\
 1;33m \
\033[40m\033[1;33m  Amarillo      \033[0m"
echo -e "\033[47m\033[34m  Azul          \033[0m\
   34m \
\033[40m\033[34m  Azul          \033[0m"
echo -e "\033[47m\033[1;34m  Azul Claro    \033[0m\
 1;34m \
\033[40m\033[1;34m  Azul Claro    \033[0m"
echo -e "\033[47m\033[35m  Púrpura       \033[0m\
   35m \
\033[40m\033[35m  Purpura       \033[0m"
```

```
echo -e "\033[47m\033[1;35m  Rosa          \033[0m\
 1;35m \
\033[40m\033[1;35m  Rosa          \033[0m"
echo -e "\033[47m\033[36m  Cyan          \033[0m\
   36m \
\033[40m\033[36m  Cyan          \033[0m"
echo -e "\033[47m\033[1;36m  Cyan Claro    \033[0m\
 1;36m \
\033[40m\033[1;36m  Cyan Claro    \033[0m"
```

5.2 Movimiento del cursor

Las secuencias de escape ANSI permiten mover el cursor por la pantalla a voluntad. Esto es más útil para interfaces de usuario a pantalla completa generados por shell scripts, pero también se pueden usar en prompts. Las secuencias de escape de movimientos son las siguientes:

```
- Posicionar el cursor:
  \033[<L>;<C>H
  pone el cursor en la linea L, columna C.
- Mover el cursor arriba N lineas:
  \033[<N>A
- Mover el cursor abajo N lineas:
  \033[<N>B
- Mover el cursor hacia adelante N columnas:
  \033[<N>C
- Mover el cursor hacia atras N columnas:
  \033[<N>D
- Guardar la posicion del cursor:
  \033[s
- Restaurar la posicion del cursor:
  \033[u
```

Los dos últimos códigos no están presentes en muchos emuladores de terminal. Los únicos que conozco que los soportan son **xterm** y **nxterm**, a pesar de que la mayoría de los emuladores de terminal están basados en el código de **xterm**. Por lo que yo sé, ni **rxvt**, **kvt** ni **xiterm** ni **Eterm** no soportan esto. La consola sí lo soporta.

Pruebe a poner la siguiente línea de código en el prompt (está más claro lo que hace si el prompt está bastantes líneas más abajo que el tope superior de la terminal)

```
echo -en "\033[7A\033[1;35m BASH \033[7B\033[6D"
```

Esto debería mover el cursor 7 líneas hacia arriba de la pantalla, escribir la palabra `BASH`, y volver a su sitio habitual en el prompt. Esto no es un prompt, es solo una demostración de movimiento del cursor por la pantalla, usando color para enfatizar lo que se ha hecho.

Salve lo siguiente en un fichero llamado `clock`: [2]

```
#!/bin/bash
```

[2]N.del T.:
para que funcione bien, he tenido que poner `let prompt_x=$COLUMNS-7`, en lugar de la línea original

```
function prompt_command {
let prompt_x=$COLUMNS-5
}

PROMPT_COMMAND=prompt_command

function clock {
local       BLUE="\[\033[0;34m\]"
local        RED="\[\033[0;31m\]"
local  LIGHT_RED="\[\033[1;31m\]"
local      WHITE="\[\033[1;37m\]"
local  NO_COLOUR="\[\033[0m\]"
case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="${TITLEBAR}\
\[\033[s\033[1;\$(echo -n \${prompt_x})H\]\
$BLUE[$LIGHT_RED\$(date +%H%M)$BLUE]\[\033[u\033[1A\]
$BLUE[$LIGHT_RED\u@\h:\w$BLUE]\
$WHITE\$$NO_COLOUR "
PS2='> '
PS4='+ '
}
```

Este prompt es bastante sencillo, excepto por el hecho de que mantiene un reloj en la esquina superior derecha de la pantalla (incluso aunque se varíe de tamaño el terminal). Esto NO funcionará en los emuladores de terminal que he mencionado que no aceptan guardar y recuperar la posición del cursor. Si se intenta ejecutar este prompt en cualquiera de esos terminales, el reloj aparecerá correctamente, pero el prompt quedará encajado en la segunda línea del terminal.

Véase la sección 10.5 (Prompt con Reloj Elegante e Inútil) para un uso más extensivo de estos códigos.

5.3 Movimiento del cursor con `tput`

Como ocurre con muchas cosas en unix, hay más de una forma de conseguir los mismos objetivos. Una utilidad llamada `tput` puede también usarse para mover el cursor por la pantalla, o devolver información acerca del estado del terminal. `tput` es menos flexible que las secuencias de escape ANSI para el posicionamiento del cursor: sólo se puede mover el cursor a una posición absoluta, no se puede mover con relación a la posición actual. Yo no uso `tput`, así que no voy a explicarlo en detalle. Consulte la página del manual y sabrá tanto como yo de `tput`.

6 Caracteres especiales: secuencias de escape octales

Aparte de los caracteres que se pueden teclear mediante un teclado, hay muchos otros que se pueden mostrar por la pantalla. He creado un script que permite comprobar qué tiene disponible el tipo que esté usando. El comando principal a usar para utilizar estos caracteres es `echo -e`. La opción `-e` le indica a `echo` que habilite la interpretación de caracteres escapados mediante contrabarra. Lo que aparezca a partir de un 200-400 octal será muy diferente con un tipo VGA de lo que aparezca con un tipo estándar linux. Queda avisado de que algunas de estas secuencias de escape tienen extraños efectos en el terminal, y no he intentado evitarlos. Los caracteres de dibujos de líneas y bloques (que nos resultan tan familiares a los usuarios de *WordPerfect*) utilizados masivamente en el proyecto *Bashprompt* están entre el `260` y `337` octal.

```
#!/bin/bash

#   Script: escgen

function usage {
   echo -e "\033[1;34mescgen\033[0m <valor_inferior> [<valor_superior>]"
   echo "   Generador de secuencias de escape octales: imprime todas las"
   echo "   secuencias de escape contenidas entre los valores menor y"
   echo "   mayor. Si no se proporciona el segundo valor, se imprimen"
   echo "   8 caracteres."
   echo "   1998 - Giles Orr, sin garantía."
   exit 1
}

if [ "$#" -eq "0" ]
then
   echo -e "\033[1;31mPor favor incluya uno o dos valores.\033[0m"
   usage
fi
let lower_val=${1}
if [ "$#" -eq "1" ]
then
   # Si no hay dos valores, sacar 8 caracteres
   upper_val=$(echo -e "obase=8 \n ibase=8 \n $lower_val+10 \n quit" | bc)
else
   let upper_val=${2}
fi
if [ "$#" -gt "2" ]
then
   echo -e "\033[1;31mPor favor, incluya dos valores.\033[0m"
   echo
   usage
fi
if [ "${lower_val}" -gt "${upper_val}" ]
then
   echo -e "\033[1;31m${lower_val} es mayor que ${upper_val}."
   echo
   usage
fi
if [ "${upper_val}" -gt "777" ]
   then
   echo -e "\033[1;31mLos valores no pueden superar 777.\033[0m"
   echo
   usage
fi

let i=$lower_val
let line_count=1
let limit=$upper_val
```

```
while [ "$i" -lt "$limit" ]
do
   octal_escape="\\$i"
   echo -en "$i:'$octal_escape' "
   if [ "$line_count" -gt "7" ]
   then
      echo
      #   Put a hard return in.
      let line_count=0
   fi
   let i=$(echo -e "obase=8 \n ibase=8 \n $i+1 \n quit" | bc)
   let line_count=$line_count+1
done
echo
```

También se puede usar `xfd` para mostrar todos los caracteres de un tipo X, mediante el comando `xfd -fn <tipo>`. Pinchando sobre un caracter determinado se puede obtener mucha información sobre él, incluyendo su valor octal. El script de arriba puede resultar útil en la consola, y en el caso de que no se esté seguro acerca del nombre del tipo de letra.

7 El paquete Bash Prompt

7.1 Disponibilidad

El paquete `Bash Prompt` está disponible en *http://bash.current.nu*, y es el resultado del trabajo de varias personas coordinadas por Rob Current (aka[3] *BadLandZ*). El paquete se encuentra en sus primeras betas, pero proporciona una manera simple de usar múltiples prompts (o temas), permitiendo poner prompts para los shells de ingreso (login shells), y para los subshells (es decir, poner cadenas `PS1` en `.bash_profile` y `.bashrc`).

La mayoría de los temas usan caracteres VGA extendidos, así que se ven mal a menos que se usen fuentes VGA (que no viene por defecto en la mayoría de los sistemas).

7.2 Cambio de fuentes en una `xterm`

Para usar algunos de los prompts más bonitos del paquete `Bash Prompt`, es necesario obtener e instalar fuentes que soporten el conjunto de caracteres esperado por los prompts. Se conocen como *fuentes VGA*, pero no tengo clara la distinción entre estos y las que Linux suele incluir, aunque claramente soportan diferentes conjuntos de caracteres.

Las `xterm` estándar soportan un alfabeto extendido, incluyendo muchas letras con tildes. En las fuentes VGA todo esto se reemplaza con caracteres gráficos (líneas, puntos, bloques...). Si alguien puede explicar esto con más detalle, que se ponga en contacto conmigo e incluiré la explicación aquí.

[4]

Obtener e instalar estas fuentes es de alguna forma un proceso relacionado. Primero hay que conseguir lo(s) tipo(s). Después asegurarse de que son ficheros `.pcf` o `.pcf.gz`. Si son ficheros `.bdf` viene

[3] N del T.:
«**A**lso **K**nown **A**s», *conocido también como ..., alias ....*

[4] N. del T.:
Se puede modificar un tipo determinado para que tenga caracteres gráficos y además incluya vocales acentuadas y eñes, para que resulte útil a un usuario hispanoamericano. Esto lo he hecho yo en mi sistema

bien el comando `bdftopcf` (mirar la página del manual). Hay que colocar estos ficheros en el directorio `/usr/X11R6/lib/X11/fonts/misc` (este es el lugar correcto para *RedHat 5.1* y *Slackware 3.4* [5], aunque podrían variar en otras distribuciones). Desde el directorio en cuestión hay que ejecutar el comando `mkfontdir`, seguido de `xset fp rehash`. En ocasiones es una buena idea editar el fichero `fonts.alias` del mismo directorio y crear alias más cortos para los tipos.

Para usar los nuevos tipos, hay que lanzar el emulador de terminal deseado con el comando apropiado, que se puede encontrar bien en la página del manual correspondiente o mediante la opción `-help` de línea de comando. En los emuladores de terminal más populares se usan así:

```
xterm -font <tipo>
```

o

```
xterm -fn <tipo> -fb <tipo-negrita>
Eterm -F <tipo>
rxvt -fn <tipo>
```

Hay tipos VGA disponibles en la página de Tipos ANSI de Stumpy en *http://home.earthlink.net/~us5zahns/enl/ansifont.html* (de la que he tomado mucho para escribir este documento).

8 Carga de un prompt diferente

8.1 Carga de un prompt diferente posterior

Las explicaciones en este COMO han mostrado cómo crear las variables de entorno `PS1`, o cómo incorporar las cadenas `PS1` y `PS2` a funciones que podían ser llamadas por `~/.bashrc` o como un tema por el paquete `Bash Prompt`.

Mediante el paquete `Bash Prompt`, basta con escribir `bashprompt -i` para ver una lista de temas disponibles. Para poner un prompt a los shells de ingreso (*login shells*) futuros (sobre todo la consola, pero también `telnet` y `Xterms`, dependiendo de cómo estén configuradas las `Xterms`), se hace con `bashprompt -l tema`. `bashprompt` entonces modifica el `~/.bash_profile` para llamar al tema seleccionado al arrancar. Para poner un prompt a subshells futuros (normalmente `Xterms`, `rxvt`, etc.), se hace con `bashprompt -s tema`, y `bashprompt` modifica el `~/.bashrc` para llama al tema apropiado al iniciarse.

Vea 2.6 (Valor permanente de las cadenas PS?) la nota de Johan Kullstam acerca de la importancia de poner las cadenas `PS?` en `~/.bashrc`

8.2 Carga inmediata de un prompt diferente

Se puede cambiar el prompt en el terminal actual (usando la función de ejemplo `elite` de arriba) escribiendo `source elite` seguido de `elite` (suponiendo que el fichero de la función `elite` se encuentre en el directorio de trabajo). Esto es engorroso, y deja una función extra (elite) en el espacio de entorno –si quiere limpiar el entorno, se hace con `unset elite`–. Esto parece un buen candidato para un pequeño shell script, pero un script no funciona aquí porque no puede cambiar el entorno del shell actual: solo puede cambiar el entorno del subshell en el que se ejecuta. En cuanto termina el script, el subshell desaparece y con él los cambios hechos al entorno.

[5] N. del T.:
y para la *RedHat 6.0*.

Algo que si puede cambiar el entorno del shell actual son las funciones de entorno. El paquete `Bash Prompt` coloca una función llamada `callbashprompt` en el entorno, y, mientras no se documente, no puede usarse para cargar ningún tema de `bashprompt` sobre la marcha. Mira en el directorio de temas que instala (el tema que se llama debe estar allí), hace un `source` de la función solicitada, carga la función y luego la elimina, de forma que mantiene el entorno limpio. `callbashprompt` no fue pensada para usarse de este modo, y no tiene control de errores, pero si se tiene esto en cuenta, funciona bastante bien.

9 Prompt dinámico con color según la carga del sistema

9.1 Un ejemplo de «prueba de concepto»

Esto es una «prueba de concepto» más que un prompt bonito: cambio de colores en el prompt dinámicamente. En este ejemplo, el color del nombre del host cambia dependiendo de la carga (a modo de aviso).

```
#!/bin/bash
#   "hostloadcolour" - 17 Octubre 98, by Giles
#
#   La idea aqui es cambiar el color del nombre del host en el prompt
#   dependiendo de un valor de carga que sirve de umbral.

# THRESHOLD_LOAD es el valor de la carga en un minuto
# (multiplicado por 100) al cual se desea que el prompt
# cambie de COLOUR_LOW a COLOUR_HIGH

THRESHOLD_LOAD=200
COLOUR_LOW='1;34'
          # light blue
COLOUR_HIGH='1;31'
           # light red

function prompt_command {
ONE=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\)/\1/" -e "s/ //g")
# Aparentemente "scale" en bc no se aplica a las multiplicaciones
# pero si a las divisiones
ONEHUNDRED=$(echo -e "scale=0 \n $ONE/0.01 \nquit \n" | bc)
if [ $ONEHUNDRED -gt $THRESHOLD_LOAD ]
then
    HOST_COLOUR=$COLOUR_HIGH
        # Light Red
else
    HOST_COLOUR=$COLOUR_LOW
        # Light Blue
fi
}

function hostloadcolour {

PROMPT_COMMAND=prompt_command
PS1="[$(date +%H%M)][\u@\[\033[\$(echo -n \$HOST_COLOUR)m\]\h\[\033[0;37m\]:\w]$ "
}
```

Mediante su editor favorito, salve esto en un fichero llamado `hostloadcolour`. Si tiene instalado el paquete `Bash Prompt`, funcionará como un tema. Si no lo tiene, escriba `source hostloadcolour` seguido de `hostloadcolour`. De cualquiera de las dos formas, `prompt_commad` se convierte en una función del entorno. Si examina el código, notará que los colores (`$COLOUR_HIGH y $COLOUR_LOW`) se ponen mediante un código parcial de color, es decir, `1;34` en lugar de `\[\033[1;34m\]`, cosa que hubiera preferido, pero no he sido capaz de que funcione con el código completo. Le agradecería que me avisara si lo consigue.

10 Prompt de ejemplo

10.1 Un prompt «ligero»

```
function proml {
local BLUE="\[\033[0;34m\]"
local RED="\[\033[0;31m\]"
local LIGHT_RED="\[\033[1;31m\]"
local WHITE="\[\033[1;37m\]"
local NO_COLOUR="\[\033[0m\]"
case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="${TITLEBAR}\
$BLUE[$RED\$(date +%H%M)$BLUE]\
$BLUE[$LIGHT_RED\u@\h:\w$BLUE]\
$WHITE\$$NO_COLOUR "
PS2='> '
PS4='+ '
}
```

10.2 Tema elite de Bashprompt

Es necesaria una fuente VGA

```
# Created by KrON from windowmaker on IRC
# Changed by Spidey 08/06
function elite {
PS1="\[\033[31m\]\332\304\[\033[34m\](\[\033[31m\]\u\[\033[34m\]@\[\033[31m\]\h\
\[\033[34m\])\[\033[31m\]-\[\033[34m\](\[\033[31m\]\$(date +%I:%M%P)\
\[\033[34m\]-:-\[\033[31m\]\$(date +%m)\[\033[34m\033[31m\]/\$(date +%d)\
\[\033[34m\])\[\033[31m\]\304-\[\033[34m]\\371\[\033[31m\]-\371\371\
\[\033[34m\]\372\n\[\033[31m\]\300\304\[\033[34m\](\[\033[31m\]\W\[\033[34m\])\
\[\033[31m\]\304\371\[\033[34m\]\372\[\033[00m\]"
PS2="> "
}
```

10.3 Prompt de usuario avanzado

Este es el prompt que yo uso, pero se nota un cierto retraso en la aparición del prompt en un PII-400 monousuario, así que no lo recomendaría para un P-100 multiusuario... Tómelo como una idea, más que como un prompt práctico.

```
#!/bin/bash
#----------------------------------------------------------------------
#       POWER USER PROMPT "pprom2"
#----------------------------------------------------------------------
#
#   Created August 98, Last Modified 9 November 98 by Giles
#
#  Problema: cuando la carga baja, dice "1.35down-.08". Hay que deshacerse
#  del negativo

function prompt_command
{
#   Crea la variable TotalMeg: la suma de los tamagnos de los ficheros
#   visibles del directorio actual
local TotalBytes=0
for Bytes in $(ls -l | grep "^-" | cut -c30-41)
do
    let TotalBytes=$TotalBytes+$Bytes
done
TotalMeg=$(echo -e "scale=3 \nx=$TotalBytes/1048576\n if (x<1)
      {print \"0\"} \n print x \nquit" | bc)

#      Esto se usa para calcular el diferencial en los valores
#      de carga proporcionados por el comando "uptime". "uptime"
#      proporciona medias de carga en 1, 5, y 15 minutos.
#
local one=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\)/\1/" -e "s/ //g")
local five=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\).*/\2/" -e "s/ //g")
local diff1_5=$(echo -e "scale = scale ($one) \nx=$one - $five\n if (x>0)
      {print \"up\"} else {print \"down\"}\n print x \nquit \n" | bc)
loaddiff="$(echo -n "${one}${diff1_5}")"

#   Cuenta los ficheros visibles:
let files=$(ls -l | grep "^-" | wc -l | tr -d " ")
let hiddenfiles=$(ls -l -d .* | grep "^-" | wc -l | tr -d " ")
let executables=$(ls -l | grep ^-..x | wc -l | tr -d " ")
let directories=$(ls -l | grep "^d" | wc -l | tr -d " ")
let hiddendirectories=$(ls -l -d .* | grep "^d" | wc -l | tr -d " ")-2
let linktemp=$(ls -l | grep "^l" | wc -l | tr -d " ")
if [ "$linktemp" -eq "0" ]
then
    links=""
else
    links=" ${linktemp}l"
fi
unset linktemp
let devicetemp=$(ls -l | grep "^[bc]" | wc -l | tr -d " ")
if [ "$devicetemp" -eq "0" ]
then
    devices=""
else
    devices=" ${devicetemp}bc"
fi
```

```
unset devicetemp

}

PROMPT_COMMAND=prompt_command

function pprom2 {

local        BLUE="\[\033[0;34m\]"
local  LIGHT_GRAY="\[\033[0;37m\]"
local LIGHT_GREEN="\[\033[1;32m\]"
local  LIGHT_BLUE="\[\033[1;34m\]"
local  LIGHT_CYAN="\[\033[1;36m\]"
local      YELLOW="\[\033[1;33m\]"
local       WHITE="\[\033[1;37m\]"
local         RED="\[\033[0;31m\]"
local   NO_COLOUR="\[\033[0m\]"

case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="$TITLEBAR\
$BLUE[$RED\$(date +%H%M)$BLUE]\
$BLUE[$RED\u@\h$BLUE]\
$BLUE[\
$LIGHT_GRAY\${files}.\${hiddenfiles}-\
$LIGHT_GREEN\${executables}x \
$LIGHT_GRAY(\${TotalMeg}Mb) \
$LIGHT_BLUE\${directories}.\
\${hiddendirectories}d\
$LIGHT_CYAN\${links}\
$YELLOW\${devices}\
$BLUE]\
$BLUE[${WHITE}\${loaddiff}$BLUE]\
$BLUE[\
$WHITE\$(ps ax | wc -l | sed -e \"s: ::g\")proc\
$BLUE]\
\n\
$BLUE[$RED\$PWD$BLUE]\
$WHITE\$\
\
$NO_COLOUR "
PS2='> '
PS4='+ '
}
```

10.4 Un prompt con la anchura del terminal

Un amigo se quejó de que no le gustaba tener un prompt que cambiara su tamaño debido a que tenía un `$PWD`, así que escribí este prompt que ajusta su tamaño a la anchura exacta del terminal, con el directorio de trabajo en la línea superior.

```
#!/bin/bash

#   termwide prompt
#      by Giles - created 2 November 98
#
#  La idea aqui es tener la linea superior de un prompt de 2 lineas
#  siempre de la misma anchura que el terminal. Esto se consigue
#  calculando la anchura de los elementos de texto, y rellenando
#  o truncando $PWD
#

function prompt_command {

TERMWIDTH=${COLUMNS}

#   Calcula la anchura del prompt:

hostnam=$(echo -n $HOSTNAME | sed -e "s/[\.].*//")
#   "whoami" y "pwd" incluyen un caracter de nueva línea al final
usernam=$(whoami)
let usersize=$(echo -n $usernam | wc -c | tr -d " ")
newPWD="${PWD}"
let pwdsize=$(echo -n ${newPWD} | wc -c | tr -d " ")
#   Agnadir los accesorios debajo...
let promptsize=$(echo -n "--(${usernam}@${hostnam})---(${PWD})--" \
                 | wc -c | tr -d " ")
let fillsize=${TERMWIDTH}-${promptsize}
fill=""
while [ "$fillsize" -gt "0" ]
do
   fill="${fill}-"
   let fillsize=${fillsize}-1
done

if [ "$fillsize" -lt "0" ]
then
   let cut=3-${fillsize}
   sedvar=""
   while [ "$cut" -gt "0" ]
   do
      sedvar="${sedvar}."
      let cut=${cut}-1
   done
   newPWD="...$(echo -n $PWD | sed -e "s/\(^${sedvar}\)\(.*\)/\2/")"
fi
}

PROMPT_COMMAND=prompt_command
```

```
function termwide {

local GRAY="\[\033[1;30m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
local WHITE="\[\033[1;37m\]"
local NO_COLOUR="\[\033[0m\]"

local LIGHT_BLUE="\[\033[1;34m\]"
local YELLOW="\[\033[1;33m\]"

case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="$TITLEBAR\
$YELLOW-$LIGHT_BLUE-(\
$YELLOW\${usernam}$LIGHT_BLUE@$YELLOW\${hostnam}\
${LIGHT_BLUE})-${YELLOW}-\${fill}${LIGHT_BLUE}-(\
$YELLOW\${newPWD}\
$LIGHT_BLUE)-$YELLOW-\
\n\
$YELLOW-$LIGHT_BLUE-(\
$YELLOW\$(date +%H%M)$LIGHT_BLUE:$YELLOW\$(date \"+%a,%d %b %y\")\
$LIGHT_BLUE:$WHITE\$$LIGHT_BLUE)-\
$YELLOW-\
$NO_COLOUR "

PS2="$LIGHT_BLUE-$YELLOW-$YELLOW-$NO_COLOUR "

}
```

10.5 Prompt con Reloj elegante e inútil

Este es posiblemente el prompt más atractivo (e inútil) que he creado. Debido a que muchos emuladores de terminal X no implementan las funciones de salvar y restaurar la posición del cursor, la alternativa cuando se sitúa un reloj en la esquina superior derecha es anclar el cursor a la parte baja del terminal. Esto se basa en la idea del prompt de la anchura del terminal anterior, con el dibujo de una línea desde el prompt hasta el reloj. Es necesaria una fuente VGA.

Nota: hay una sustitución en este código, que puede que no se muestre bien al pasar de SGML a otros formatos: he tenido que sustituir el carácter de pantalla pro el código \304; normalmente habría incluido la secuencia \304, pero ha sido necesaria la sustitución en este caso:

```
#!/bin/bash

#   Este prompt requiere una fuente VGA. El prompt se ancla a la parte
#   baja del terminal, rellena la anchura del terminal, y dibuja una linea
#   hacia arriba en la parte derecha del terminal hasta unirse a un reloj
#   en la esquina superior derecha del terminal
```

```
function prompt_command {
#   Calcula la anchura del prompt:
hostnam=$(echo -n $HOSTNAME | sed -e "s/[\.].*//")
#   "whoami" y "pwd" incluyen un caracter de nueva línea al final
usernam=$(whoami)
newPWD="${PWD}"
#   Se agnaden todos los accesorios
let promptsize=$(echo -n "--(${usernam}@${hostnam})---(${PWD})-----" \
                 | wc -c | tr -d " ")
#   Adivina cuanto agnadir entre user@host y PWD (o cuanto quitar a PWD)
let fillsize=${COLUMNS}-${promptsize}
fill=""
#   si el prompt no es tan ancho como el terminal, lo relleno
while [ "$fillsize" -gt "0" ]
do
   fill="${fill}Ä"
   # La A con umlaut (aparecera como un guión largo si se utiliza
   # una fuente VGA) es \304, pero la corté y pegué aquí porque
   # Bash sólo hace una sustitución, que en este caso es hacer que
   # $fill aparezca en el prompt.
   let fillsize=${fillsize}-1
done
#   Trunco $PWD por la derecha si el prompt es mas ancho que el terminal:
if [ "$fillsize" -lt "0" ]
then
   let cutt=3-${fillsize}
   sedvar=""
   while [ "$cutt" -gt "0" ]
   do
      sedvar="${sedvar}."
      let cutt=${cutt}-1
   done
   newPWD="...$(echo -n $PWD | sed -e "s/\(^${sedvar}\)\(.*\)/\2/")"
fi
#
#   Creo el reloj y la barra que sube hasta el
#
local LIGHT_BLUE="\033[1;34m"
local     YELLOW="\033[1;33m"
#   Posicion del cursor para dibujar el reloj:
echo -en "\033[2;$((${COLUMNS}-9))H"
echo -en "$LIGHT_BLUE($YELLOW$(date +%H%M)$LIGHT_BLUE)\304$YELLOW\304\304\277"
local i=${LINES}
echo -en "\033[2;${COLUMNS}H"
#   Dibujo barras verticales:
while [ $i -ge 4 ]
do
   echo -en "\033[$(($i-1));${COLUMNS}H\263"
   let i=$i-1
done

let prompt_line=${LINES}-1
#   Esto es necesario porque hacer \${LINES} dentro de una función
#   matematica de Bash (es decir $(()) ) no parece funcionar.
```

```
}

PROMPT_COMMAND=prompt_command

function clock3 {
local LIGHT_BLUE="\[\033[1;34m\]"
local     YELLOW="\[\033[1;33m\]"
local      WHITE="\[\033[1;37m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
local  NO_COLOUR="\[\033[0m\]"

case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="$TITLEBAR\
\[\033[\${prompt_line};0H\]
$YELLOW\332$LIGHT_BLUE\304(\
$YELLOW\${usernam}$LIGHT_BLUE@$YELLOW\${hostnam}\
${LIGHT_BLUE})\304${YELLOW}\304\${fill}${LIGHT_BLUE}\304(\
$YELLOW\${newPWD}\
$LIGHT_BLUE)\304$YELLOW\304\304\304\331\
\n\
$YELLOW\300$LIGHT_BLUE\304(\
$YELLOW\$(date \"+%a,%d %b %y\")\
$LIGHT_BLUE:$WHITE\$$LIGHT_BLUE)\304\
$YELLOW\304\
$LIGHT_GRAY "

PS2="$LIGHT_BLUE\304$YELLOW\304$YELLOW\304$NO_COLOUR "

}
```

11 Anexo: El INSFLUG

El *INSFLUG* forma parte del grupo internacional *Linux Documentation Project*, encargándose de las traducciones al castellano de los Howtos (Comos), así como la producción de documentos originales en aquellos casos en los que no existe análogo en inglés.

En el **INSFLUG** se orienta preferentemente a la traducción de documentos breves, como los *COMOs* y *PUFs* (**P**reguntas de **U**so **F**recuente, las *FAQs*. `:)`), etc.

Diríjase a la sede del INSFLUG para más información al respecto.

En la sede del INSFLUG encontrará siempre las **últimas** versiones de las traducciones «oficiales»: *www.insflug.org*. Asegúrese de comprobar cuál es la última versión disponible en el Insflug antes de bajar un documento de un servidor réplica.

Además, cuenta con un sistema interactivo de gestión de fe de erratas y sugerencias en línea, motor de búsqueda específico, y más servicios que estamos trabajando incesantemente para añadir.

Se proporcionará también una lista de los servidores réplica (*mirror*) del Insflug más cercanos a Vd., e información relativa a otros recursos en castellano.

En *http://www.insflug.org/insflug/creditos.php3* cuenta con una detallada relación de las personas que hacen posible tanto esto como las traducciones.

¡Diríjase a *http://www.insflug.org/colaboracion/index.php3* si desea unirse a nosotros!.

Francisco José Montilla, *pacopepe@insflug.org*.