

# Ampliando explicaciones sobre el prompt que a mí me gusta

**Por shicefgo.** "Cuando no sepas nada acerca de algo, escribe un libro sobre ello".

Liberada bajo licencia



<http://creativecommons.org/licenses/by-nc-sa/2.5/>

## 1. Introducción

En uno de mis "posts", o envíos, a los foros de *Fentlinux* (<http://www.fentlinux.com>), se me ocurrió poner el código del prompt que uso en la terminal de texto de linux, pero como aquello lo hice más bien de prisa y corriendo, lo mismo no está de más que intente explicarlo algo mejor.

Antes de entrar en materia, considero necesario decir que la única distribución de Linux que conozco lo suficiente como para atreverme a hablar de ella es *Debian*, así que pido disculpas si algo de lo que digo no funciona exactamente igual en otra, aunque procuraré referirme, en la medida de mis posibilidades, a cuestiones generales que deberían funcionar de la misma forma en cualquier sistema Linux o, mejor dicho, GNU/Linux.

También deseo aclarar que soy autodidacta de toda la vida, lo que quiere decir que podré conocer de modo aceptable algunos aspectos de algunas materias, pero que también puedo tener lagunas (y es seguro que las tengo) más o menos como el Pacífico, por tanto, no pretendo enseñar nada a nadie al escribir esto, si no más bien compartir lo aprendido como una forma de agradecer el esfuerzo de la comunidad del software libre, y para animar a otros con más conocimientos a hacer lo mismo. Si yo he podido, seguro que cualquiera puede, es cuestión de ganas y voluntad pero, sobre todo, es cuestión de que algo que te gusta también te divierta, y de apasionarse por profundizar en el conocimiento de las cosas.

Por supuesto que esto no va a ser nada parecido a un manual. Más bien pienso que pueda compararse a una especie de recordatorio compartido, algo así como unos apuntes sobre el prompt puestos a disposición de todo aquél a quien le pueda interesar. Bueno, pues a ver de lo que me acuerdo, porque ya hace algún tiempo que no trasteo con scripts de shell.

## **2. Lo que son las cosas (o como nos vamos a entender)**

### **2.1. Un teclado de ordenador**

Es lo más sencillo de utilizar para hacer cosas como el prompt del que hablamos, ya que escribir con el ratón es bastante difícil, pero si alguien quiere agenciarse un programa de esos que dibujan un teclado en la pantalla y es capaz de capturar los clicks en las letras y ponerlos donde quiera, pues adelante: si funciona, estupendo.

### **2.2. Un terminal**

Es una pantalla negra, con un indicador a la izquierda, a partir del cual se pueden escribir cosas con el teclado (aquí no funcionaría el sistema anterior para el ratón). He leído por ahí algo que, más o menos, se podría traducir así: "Si usted sienta a un mono delante de un teclado de ordenador, probablemente lo primero que escribirá será un comando unix" (no creo que haga falta comentar nada acerca del paralelismo unix/linux, pero dicho queda, por si acaso). Esta frase, a mi entender, es una forma retorcida de decir que los comandos unix no tienen demasiada lógica en su denominación. Además, también me dijo alguien una vez que casi cualquier combinación de dos letras sería también un comando unix. Pues bien, lo único que el ordenador entenderá de lo que escribamos con el teclado en la terminal, serán comandos, programas ejecutables, o scripts de shell, también ejecutables. Los programas ejecutables los dividiremos en dos tipos: ejecutables en terminal y ejecutables en modo gráfico. Si escribimos el nombre de un programa para entorno gráfico en un terminal, no se ejecutará y aparecerá un mensaje diciendo algo acerca de que no hay display.

### **2.3. Una ventana de terminal**

Es el terminal de antes, pero situado en una ventana, en el entorno gráfico. Aquí se pueden ejecutar tanto los programas puramente de terminal como los concebidos para funcionar en un contexto gráfico.

### **2.4. Un prompt**

Es el indicador a la izquierda en el terminal del que hablábamos antes, a partir del cual se pueden escribir los nombres de comandos, programas y scripts de shell.

### **2.5. Una shell**

Para quien realmente no tenga idea de lo que es una shell, digamos que piense en un programa que se encarga de interpretar las órdenes que le llegan, ya sea directamente del teclado o a través de un script, aunque tal vez quedaría mejor explicado como "un entorno de ejecución". Cuando escribimos algo

como: **alias ls='ls --color=auto'**, estamos invocando a la shell, ya que no hay ningún programa llamado "alias" grabado en nuestros discos. En cambio, cuando escribimos: **ls /usr/bin/g\***, la shell se encarga de que al ejecutarse el comando **ls** se expanda el carácter "\*", con lo cual obtenemos un listado de todo lo que hay en /usr/bin/ cuyo nombre comience con la letra 'g'.

Por lo que yo sé, la shell más utilizada en linux es bash.

## **2.6. Un ejecutable**

Es lo que todo el mundo entiende por "un programa de ordenador".

## **2.7. Un comando**

Es un programa, pero con la característica especial de que pertenece al sistema operativo. Cada sistema operativo tiene sus propios nombres para los comandos: el "copy" de MS-DOS es el "cp" de linux, por ejemplo (bueno, no exactamente, pero para entendernos vale).

## **2.8. Un binario**

Es un programa de ordenador compilado y enlazado, es decir, que se ejecuta a partir de código entendible directamente por la máquina.

## **2.9. Un script de shell (o simplemente script)**

Es un programa de ordenador que procesa instrucciones entendibles por la shell, y que se ejecuta a partir de un archivo de texto, en el que están escritas "en cristiano" (quiero decir que no en binario) las instrucciones, normalmente comandos, siguiendo las reglas de sintaxis de la shell, naturalmente.

## **2.10. Un sistema operativo**

Para intentar explicarlo de una forma sencilla, se puede decir que un sistema operativo es un conjunto de programas que hacen posible que un humano pueda comunicarse con un ordenador. La parte más importante de un sistema operativo es el núcleo, o kernel. Linux es el nombre de un kernel que se ha hecho extensible a todo el sistema operativo.

## 2.11. Un editor de texto

Es un programa que nos va a permitir crear y modificar archivos con el contenido que nosotros queramos, utilizando los caracteres que se pueden obtener con el teclado directamente, es decir, que los archivos contendrán lo que escribamos en ellos, sin ningún tipo de codificación añadida.

Vamos a llamar *procesador* de texto al programa que permite visualizar en la pantalla el archivo tal y como quedará impreso, con sus párrafos, tipos de letra, gráficos... etc. Esto se conoce como WYSIWYG, que significa "lo que ves es lo que obtienes". Lógicamente, para conocer los atributos de cada carácter escrito (negrita, cursiva, color, tipo de letra, subrayado... etc.) los procesadores de texto tienen que grabar los archivos con algún tipo de codificación que permita definir todos los atributos posibles en el texto. Open Office lo hace utilizando el xml. Se puede descomprimir con el programa **gunzip**, por ejemplo, cualquier archivo de Open Office, y veremos que aparecen varios archivos nuevos, creo que unos cuatro, que están escritos en texto llano con marcas xml. Uno de esos archivos contiene el texto introducido por nosotros, el cual podremos reconocer, no sin algo de paciencia, entre las numerosas etiquetas xml. El procesador de texto Microsoft Word™, y supongo que otros también lo harán de manera análoga, guarda los archivos en un formato propio de tipo binario, por lo que únicamente es posible acceder a su contenido utilizando su formato codificado, y ningún otro. Si bien para Open Office y algunos otros es posible acceder a documentos escritos con las aplicaciones de la suite de Microsoft™, esto es sólo aplicable a las versiones anteriores a la 2003. A partir de esa versión, tengo entendido que ya los nuevos archivos creados no pueden ser accedidos por otros procesadores de texto, ni siquiera por versiones anteriores del MS-Office. Aunque, según parece, Microsoft™ ha anunciado que incorporará el XML a sus herramientas. Faltaría saber en qué forma, pero esto por sí sólo ya indica que no les va quedando más remedio que pensar en utilizar los estándares.

Personalmente, sólo utilizo los procesadores tipo WYSIWYG para escribir cartas comerciales, y más bien porque no se le puede pedir a la gente que aprenda a usar para eso cosas como Docbook o LaTeX, de modo que hay que trabajar con lo que todo el mundo conoce y sabe usar. Esto lo estoy escribiendo en Docbook-xml, lo que me permite colocar una serie de etiquetas que describen cosas como: "esto es un párrafo", "esto va resaltado", "esto es código de un programa", "esto es un título de un párrafo..." y así. Luego puedo utilizar una serie de herramientas para que ese escrito se convierta al formato html, pdf, postscript, texto llano, una página tipo man, LaTeX, o también rtf, que es un formato comprensible por los procesadores de texto tipo WYSIWYG, como el Microsoft Word™ o el Open Office. No tengo que estar constantemente preocupado por poner esto en negrita, lo de más allá en cursiva, aquí va el título del párrafo con otra fuente más grande... etc., y no olvidarme del estilo adoptado en todo el documento, lo que puede ser un problemilla si éste es bastante grande. Y no digamos si son varias personas las que colaboran para su redacción, porque entonces alguien tendrá que coordinar el trabajo de todo el mundo para conseguir algo homogéneo y más o menos presentable. Pero hay algo todavía mejor, al menos para mí: Docbook está hecho en XML (o quizás sea más apropiado decir en SGML, ya que XML es un subconjunto de éste), y XML es un estándar reconocido, mientras que el formato de Microsoft™, por ejemplo, es único y exclusivo de ellos. Si algún día no puedo tener acceso al Word, no podré leer ni modificar *MI* trabajo escrito con el Word, o tendré que estar actualizándome continuamente a la versión que toque, previo pago de su importe, naturalmente. En cambio, utilizando editores de texto y marcas XML, siempre tendré acceso a mi trabajo con alguno de los múltiples editores de texto que hay disponibles.

El editor que ahora mismo utilizo, y que pienso seguir utilizando en el futuro, es emacs. Pero no porque crea que es mejor que otro, si no porque es con el que más cómodo me encuentro. He usado vim durante una buena temporada, y también conozco algo a joe y a xedit, y he llegado a mirar de reojo al nano, pero definitivamente me quedo con emacs y ya está.

Hago mención de los editores porque son las herramientas necesarias para programar, que es en definitiva de lo que se trata al hacer un script, de modo que, pensando siempre en dirigirme a quienes menos idea tengan, iré indicando la manera de hacer las cosas con emacs. Ni que decir tiene que quien quiera informarse más y mejor sobre los editores de texto disponibles, no tiene más que efectuar una búsqueda en la internet sobre la frase "editores de texto", con su buscador favorito o, si lo prefiere, buscar directamente por el nombre del editor.

## **2.12. El "código fuente" de un programa**

Son archivos en texto llano, normalmente, que contienen el programa en un formato más o menos inteligible, de acuerdo con las reglas del lenguaje de programación utilizado. El empleo del término "fuente" para referirse al código del que procede un programa, viene de la traducción de la palabra inglesa *source*, mientras que el mismo término aplicado a un tipo de letra, proviene de *font*. Rebuscando en un diccionario inglés-spanish del siglo pasado, y poniéndole algo de imaginación a la cosa, se me ocurre que *source* se ha traducido por fuente en el sentido de "origen", de "fuente primordial", mientras que *font* se traduce también por fuente pero de una manera algo más literal, ya que el diccionario incluye la acepción "fundición" como uno de los significados de *font*, de modo que aquí interpreto que el concepto de "tipo de letra", en el inglés, está asociado a la idea de su fundición en plomo en las antiguas imprentas, y se traduce a veces de forma un tanto errónea, porque la palabra "tipo" significa precisamente eso en español.

## **2.13. Un lenguaje de programación**

Pues eso mismo: un lenguaje, con su vocabulario y su sintaxis, pero creado artificialmente para poder realizar programas de ordenador. Atreviéndome mucho, podría intentar mencionar algunos elementos comunes a los lenguajes de programación:

- Un conjunto de palabras "reservadas", es decir, que no pueden utilizarse para otro propósito que no sea el previsto por el lenguaje.
- Una clasificación de las variables en tipos, como pueden ser: numérico, alfanumérico (admiten números, letras y signos, pero no se pueden realizar cálculos) y, dentro de éstos, numérico entero, numérico de coma flotante, alfanumérico de un sólo carácter, alfanumérico de varios caracteres... etc.
- Un subgrupo de las "palabras reservadas" para instrucciones de control, como pueden ser "if", "else", "do", "while", "for"... y análogas. Los operadores, como los caracteres '+', '-', '\*' (multiplicación), '/' (división), '<', '>...' etc.

Dejando aparte las posibles subdivisiones en función de la idoneidad de cada lenguaje para tareas específicas, hay dos clases fundamentales de lenguajes de programación: lenguajes compilados y lenguajes interpretados. Un lenguaje compilado produce un programa en código binario que, como ya se dijo anteriormente, es directamente ejecutable por la máquina. En cambio, el programa producido por un lenguaje interpretado tiene que ser procesado por un "intérprete" para que la máquina pueda entenderlo. A ese intérprete se le llama en ocasiones "máquina virtual", expresión que seguramente les resultará familiar a quienes hayan leído algo sobre Java. Fortran, C y C++ son ejemplos de lenguajes compilados. Java, el de la shell bash y la plataforma mono, o net, si hablamos de Microsoft™, con el C#, lo serían de lenguajes interpretados. Las diferencias entre ambas clases de lenguajes vienen a ser que, mientras los compilados son más rápidos y eficientes en su ejecución, los interpretados permiten al programador abstraerse por completo de la arquitectura del sistema donde el programa vaya a ser ejecutado. Los lenguajes compilados dependen de que el compilador haya sido portado a la arquitectura, y los interpretados dependen de que la máquina virtual que los ejecuta también haya sido portada a la misma arquitectura y, mientras en los compilados habría que compilar el programa para cada arquitectura diferente, en los interpretados no haría falta, ya que bastaría instalar el intérprete, o máquina virtual. Con los ordenadores que se están fabricando a estas alturas, el mayor rendimiento de los lenguajes compilados queda compensado por la potencia de las nuevas máquinas, así que no creo que haya diferencias apreciables a los sentidos humanos en este aspecto, de modo que la discusión casi queda reducida a una cuestión de gustos, en el caso de que se pueda elegir, y también a la facilidad de aprendizaje y comodidad de uso que cada cual encuentre en unos y otros.

Si quieres programar, no empieces preguntando qué lenguaje elegir o cuál es mejor que otro. Escoge uno que te suene bien y empieza a escribir código con él. Más adelante, cuando obtengas algo de experiencia, irás valorando pros y contras, y acabarás decantándote por el tuyo. Para aprender a programar, hay que programar lo que sea con lo que sea. Luego, cuando ya se conozca la naturaleza de los problemas a los que nos tenemos que enfrentar, será el momento de elegir las herramientas que mejor se adapten a nuestra forma de resolverlos y, por supuesto, a nuestros gustos.

## **2.14. Una función**

En el contexto de la programación, que es el que nos ocupa, una función es un trozo de programa que puede ser ejecutado desde distintos lugares, ya sea del mismo programa o de otros, para lo que, en el argot, se dice que hay que "llamarla". Las funciones de la shell bash podemos grabarlas en disco, para luego "cargarlas" en memoria RAM y ejecutarlas. "El prompt que a mí me gusta" es una función de este tipo.

## **3. Por qué he modificado el prompt**

Para aprender a hacerlo y, de paso, poder distinguir a golpe de vista en qué equipo trabajo cuando estoy conectado a más de uno, gracias a los colores.

Como acostumbro a tener abierta una ventana de terminal en otro escritorio, para compilar y hacer "ls" y

"find" de vez en cuando, mantengo a la vista la hora actual cada vez que pulso la tecla de retorno. Esto me viene bien para caer en la cuenta de que ya va siendo hora de irse a la cama cuando estoy concentrado en una tarea y el sueño no aprieta todavía demasiado (vamos, que aún puedo mantener los ojos abiertos).

También me gusta saber en qué día estoy, para poner la fecha en alguno de los archivos que informan de cuando se hacen las cosas, como sería el caso del ChangeLog, y esto cambia cada vez que llega la media noche.

Ya me he habituado a tenerlo así, y dudo que cambie el poder ver la hora y la fecha, porque suelo teclear bastante con nocturnidad, aunque sin ninguna alevosía.

Seguramente a más de uno esto le parezca una tontería, ya que es posible poner un reloj en el escritorio con un calendario al lado, o simplemente utilizar los que algunos gestores de ventanas ponen en el panel, pero es que también suelo teclear en terminales, sin entrar en el entorno gráfico, y ahí me encuentro "sólo ante el prompt".

## 4. Entrando en materia

*En este artículo* ([http://www.fentlinux.com/wiki/index.php?title=Personalizando\\_la\\_SHELL](http://www.fentlinux.com/wiki/index.php?title=Personalizando_la_SHELL)) del wiki, ZX80 explica todo lo referente a los colores, códigos y comandos a emplear en la personalización del prompt. Por mi parte, trataré de hacer inteligible la forma en que mi prompt ha sido programado y su funcionamiento.

Escribiendo **emacs** en la shell de un terminal, o en la de una ventana de terminal (y pulsando a continuación la tecla 'entrar', claro), entramos al editor de texto siempre que lo tengamos instalado, obviamente. Una vez que haya terminado de cargarse, pulsamos la tecla Control y, sin soltarla, las teclas 'x' y 'f', lo que abreviaremos así: C-x-C-f (causa el mismo efecto pulsar Control-x, liberar Control, y volver a pulsar Control-f). Si hubiera que pulsar Control más la tecla 'x', liberar la tecla Control y pulsar 'b', lo abreviaríamos así: C-x-b. C-x-b, por cierto, sirve para cambiar del archivo que estamos editando al que habíamos editado anteriormente, en el caso de que estemos trabajando con varios archivos (buffers, en la jerga de emacs), simultáneamente. Si nos damos cuenta de que nos hemos equivocado de teclas, y tenemos algo en el llamado mini-buffer (la penúltima línea del editor) que no es lo que queríamos, basta con pulsar C-g (Control-g) para anular la orden y volver al estado normal de edición. Si lo que necesitamos es deshacer el último cambio efectuado, habrá que pulsar Control-mayúsculas y la tecla del guión '-' simultáneamente. Si no liberamos Control ni mayúsculas y efectuamos repetidas pulsaciones sobre la tecla del guión, iremos deshaciendo hacia atrás hasta llegar al límite, momento en el que emacs avisará con un mensaje en el mini-buffer y, si continuáramos, volvería a rehacer lo ya deshecho, así que ojo con los bucles infinitos :-). Por supuesto que también funciona la tecla de retroceso. El sistema anterior es más bien para usarlo en casos como un copy-paste, una sustitución de texto... y cosas así. Bien: habíamos pulsado C-x-C-f, miramos el mini-buffer, y vemos algo así: ~/ , y a continuación un cursor, posiblemente parpadeante. Escribamos en esa posición el nombre que queramos darle al archivo a crear, en mi caso: super\_prompt, pero que cada cual lo llame como más le guste.

Con las siguientes líneas:

```
function super_prompt
{
}
```

di comienzo a la programación de la función que he dado en llamar, de modo casual pero no original, "super\_prompt". Tengo por costumbre escribir las llaves de apertura y de cierre al empezar, para luego colocar el código en medio, procurando de esta forma evitar el olvido de alguna llave de cierre cuando se anidan varios bloques de código, cosa bastante frecuente en C, por ejemplo. Con esas tres líneas se le está diciendo a la shell que se quiere dar el nombre de super\_prompt a un trozo de programa, que empieza tras el carácter { y termina ante el carácter }. Como los "trozos de programas" se llaman funciones, pues hay que anteponer la palabra reservada "function" al nombre de la misma para que la shell sepa por donde van las cosas.

```
function super_prompt
{

# Valores de los colores por orden alfabético.
local AMARILLO="\[\033[1;33m\"
local AZUL="\[\033[0;34m\"
local AZUL_CLARO="\[\033[1;34m\"
local BLANCO="\[\033[1;37m\"
local CYAN="\[\033[0;36m\"
local CYAN_CLARO="\[\033[1;36m\"
local GRIS="\[\033[1;30m\"
local GRIS_CLARO="\[\033[0;37m\"
local MARRON="\[\033[0;33m\"
local NEGRO="\[\033[0;30m\"
local PURPURA="\[\033[0;35m\"
local ROJO="\[\033[0;31m\"
local ROJO_CLARO="\[\033[1;31m\"
local ROSA="\[\033[1;35m\"
local VERDE="\[\033[0;32m\"
local VERDE_CLARO="\[\033[1;32m\"

... / ...
}
```

Con el anterior código, utilizamos variables para almacenar el valor de algunos colores. El carácter '#' significa que esa línea es un comentario, y no será tomada en cuenta por el intérprete de la shell (bash), y la palabra "local", que sólo puede utilizarse en una función, significa que esa variable no será accesible desde fuera de la función donde ha sido declarada. El uso de nombres clarificadores, como yo los llamo, para algunos valores, ayuda a hacer más comprensible el código que escribimos. En el pasado me ha ocurrido que, tras bastante tiempo sin tocar un programa, tener que modificarlo y perder media mañana en averiguar por qué y para qué había escrito aquello.

Una vez que hemos llamado a los colores por sus nombres, pasamos a escribir un par de líneas más:

```
... / ...
```

```
# Colores a utilizar en el prompt.  
local COLOR=$VERDE  
local COLOR_CLARO=$VERDE_CLARO  
... / ...
```

Que serán las que habrá que modificar para elegir los colores del prompt. Verde y verde claro es lo que tengo en el equipo de casa, mientras que en el trabajo tengo el amarillo y el blanco. Cuando me conecto desde casa, via ssh, para actualizar el sistema del servidor a una hora en la que no haya nadie trabajando, lo hago desde una consola de texto, no desde una ventana, y el prompt amarillo me recuerda en todo momento a qué equipo estoy conectado. Este tipo de conexiones funcionan muy bien en consola de texto, por lo que el prompt me resulta útil.

Hasta ahora, todo lo escrito puede hacerse con emacs sin mayores problemas y utilizando las teclas normales de edición: las flechas, retroceso, suprimir, etc. En el momento que queramos guardar el trabajo que llevamos hecho, tenemos dos formas: C-x-C-s y C-x-s. La primera guardará el archivo actual si más, y la segunda nos preguntará si queremos guardar cada uno de los archivos que estemos editando en ese momento. (Recordad que C-x-C-s es pulsar Control-x + la 's' sin liberar Control, y que C-x-s es pulsar Control-x, liberar Control, y pulsar s). Sigamos con otra línea de código:

```
... / ...  
# N° del terminal actual.  
local TERMINAL=$(tty | cut -d/ -f3)  
... / ...
```

Esto es también un nombre de variable "clarificador". Le llamamos "TERMINAL" al n° del terminal en el que estamos. Los terminales disponibles se encuentran en el archivo /etc/inittab, en unas líneas como las siguientes:

```
1:23:respawn:/sbin/getty 38400 tty1  
2:23:respawn:/sbin/getty 38400 tty2  
3:23:respawn:/sbin/getty 38400 tty3  
4:23:respawn:/sbin/getty 38400 tty4  
5:23:respawn:/sbin/getty 38400 tty5  
6:23:respawn:/sbin/getty 38400 tty6
```

Eso define 6 terminales diferentes que, en modo texto, son accesibles pulsando Alt-Fn (Fn = teclas de función F1 hasta F6) y, desde el modo gráfico, pulsando Control-Alt-Fn. En mi caso no es raro que tenga abiertas un par de terminales, sobre todo cuando actualizo el sistema, o cuando estoy conectado al trabajo. Podemos aumentar (o disminuir) el número de terminales si queremos, pero habremos de tener en cuenta que, normalmente, el terminal n° 7 es utilizado para contener la sesión gráfica por defecto, de ahí que, si hemos pasado del modo gráfico a un terminal pulsando Control-Alt-F2, por ejemplo, para volver al modo gráfico tengamos que pulsar Alt-F7. Ahora vamos con la explicación de la línea.

"local TERMINAL=\$( ... )". Esto significa que le queremos asignar a la variable TERMINAL el valor (representado por el '\$') de la expresión que haya entre los paréntesis. Y, dentro de los paréntesis: tty significa que nos diga cuál es el dispositivo de terminal que estamos usando. Esto podemos verlo

tecleando **tty** directamente en el terminal, pero si lo hacemos en una ventana de terminal, nos dirá algo así como: /dev/pts/0. Como el comando **tty** produce una salida, esta la redirigimos mediante la tubería (pipe) '|' al comando **cut -d/ -f3** (para explicar esto se suele emplear la expresión: "un comando toma como entrada la salida de otro"), lo que significa que, a la salida por pantalla que producirá el comando **tty**, se le aplica la ejecución del comando **cut -d/ -f3**, que viene a significar: "cortar (cut), tomando como separador de campo el carácter '/', (-d/) lo que haya en el tercer campo de la ristra de caracteres" (-f3). Conviene aclarar en este punto que, para la shell bash, el primer campo sería el borde de la pantalla (para entendernos a lo práctico), el segundo lo que haya entre la primera '/' y la segunda, y el tercero lo que haya entre la segunda '/' y la tercera, que es lo que nos interesa, si comprobamos la salida del comando **tty**, esto es, el dispositivo de terminal en el que estamos trabajando. En este punto, hacemos un C-x-C-s y, una vez salvado el arduo trabajo que llevamos realizado, tomamos un sorbito de algo que contenga café y nos disponemos a continuar. Aprovecharé para decir que para abandonar la edición con emacs basta teclear C-x-c. (Control-x, liberar Control y pulsar 'c'). Emacs se encargará de avisarnos si no hemos guardado las últimas modificaciones realizadas en los archivos editados. Con el tiempo uno termina acostumbrándose a utilizar emacs con un ojo puesto en el mini-buffer :).

Siguientes líneas de código:

```
# Si el usuario es root, el prompt termina en '#', de lo contrario en '$'.

if [ "`id -u`" -eq 0 ]; then
  local IDU='#'
else
  local IDU='$'
fi
```

Aquí tenemos la palabra reservada "if", que es un "si" condicional, los caracteres '[' y ']', que encierran la expresión a evaluar, el ';' y la palabra "then", que forman parte de la sintaxis de la shell, lo que hay que hacer si la expresión es cierta, la palabra reservada 'else', lo que hay que hacer si la expresión no es cierta, y la palabra reservada "fi", que significa "fin del código relativo a la expresión condicional". Los "if" pueden anidarse, es decir, que puede haber un "if" dentro de otro "if", en cuyo caso los "fi" indicarían el fin del "if" de su mismo nivel. Un ejemplo:

```
if [ condicion-1 ]; then
  proceso_si_condicion-1_se_cumple
  if [ condicion-2 ]; then
    proceso_si_condicion-2_se_cumple
  fi # Fin del if de condicion-2
fi # Fin del if de condicion-1
```

En el anterior código, condición-2 sólo se evaluaría si condición-1 se cumple. Veamos ahora "id -u". Esto es lo mismo que \$(id -u), digamos que la primera forma, la de las comillas inversas, sería una expresión "a la antigua". El efecto que tienen ambas formas es obtener el valor de **id -u** tal como aparecería en el terminal si lo tecleáramos. Recomiendo hacerlo, y veremos un número, que será el cero en caso de que lo hagamos como root, y otro distinto de cero si lo hacemos como usuario normal. Ese número es el que utiliza el sistema operativo para identificar al usuario. La expresión **-eq 0** significa "igual a cero", por lo que ya hemos desentrañado el significado completo de la expresión a evaluar: "if [ "id -u" -eq 0 ];" que significa: "Si la identidad del usuario es igual a cero", es decir: "si el usuario es el

root"; entonces (then), si la expresión se cumple, asignamos el carácter '#' a la variable IDU (Identificación Del Usuario) y, si el usuario no es el root, le asignamos el '\$', que son los típicos caracteres con los que finaliza el prompt y a partir de los cuales se escriben las órdenes para la shell. Es necesario dejar un espacio entre los corchetes y el comienzo y final de la expresión; esto forma parte de la sintaxis del lenguaje de programación de la shell bash, al igual que el ';' después del corchete de cierre, y la palabra "then".

-eq, -ne, -lt, -le, -gt, y -ge son operadores binarios aritméticos que devuelven verdadero si, respectivamente, el valor a su izquierda es igual a, distinto de, menor que, menor o igual a, mayor que, o mayor o igual, al valor de su derecha. Es decir: comparan números. Las cadenas se suelen comparar con los operadores == (igual a) y != (distinto de). Recomiendo una lectura de las páginas del manual de bash (**man bash**) para una mejor comprensión del uso de los operadores, ya que esto no pretende ser un tutorial del mismo. A lo sumo, y siendo bastante pretencioso, podría considerarse "una aproximación a una introducción".

Siguiente código:

```
# Definición del prompt PS1.

PS1="\
$COLOR_CLARO[\
$COLOR\u\
$COLOR_CLARO@\
$COLOR\h\
$COLOR_CLARO]-[\
$COLOR\#\/$TERMINAL\
$COLOR_CLARO]-[\
$COLOR\$(date +%H:%M)\
$COLOR_CLARO]-[\
$COLOR\$(date +%A-%d-%b-%Y)\
$COLOR_CLARO]\n\
$COLOR\w$IDU\[\033[0m\] "
```

Los caracteres '\`' (escape) al final de cada línea indican que la línea continúa y que no se tenga en cuenta el retorno de carro que le sigue. PS1 es el nombre del prompt primario, y PS2 el del prompt secundario, que es el que aparece cuando tecleamos una orden muy larga, ponemos el carácter '\`', pulsamos intro y continuamos en la línea siguiente, la cual mostrará el prompt PS2. Esto no tiene demasiado que explicar: ponemos el color y el dato que se mostrará, y listo. Los datos más relevantes que se muestran son:

\u

Nombre del usuario.

\h

El nombre del equipo (host) hasta el primer punto. Es decir, si el nombre fuese algo como: equipo.midominio.com, se vería únicamente "equipo".

```
\\#/$TERMINAL
```

El número de comandos introducidos (\\#), una barra, y el terminal en que nos encontramos, que anteriormente obtuvimos con: `TERMINAL=$(tty | cut -d/ -f3)`.

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

Obtenemos del comando **date** la hora (%H) y los minutos (%M), y los separamos por los dos puntos.

```
\\$(date +%A-%d-%b-%Y)
```

De manera análoga a la anterior, obtenemos del comando **date** el nombre del día de la semana con todas sus letras (%A), el día del mes (%d), el nombre del mes abreviado (%b) y el año con 4 dígitos (%Y), y separamos los datos con guiones ('-'). Es necesario anteponer el signo '+' al principio de la secuencia a obtener.

```
\\n
```

Un retorno de carro, que nos envía a una nueva línea. El equivalente a una pulsación de la tecla Enter, Intro, Return, Retorno, Entrar... ¿etc?

```
\\w$IDU\\[\\033[0m\\] "
```

El directorio actual (\\w), menos si es nuestro \$HOME, el carácter '#' si se es root, o el '\$' si no (el contenido de la variable \$IDU, que ya obtuvimos antes), y dejamos el terminal con su fondo predeterminado y su color de letra (\\[\\033[0m\\]), para no seguir escribiendo en el mismo color que tuviese el prompt. Al final hay un espacio extra para que el cursor no se sitúe inmediatamente a continuación del prompt, que es algo que me produce mal efecto. Las secuencias de caracteres "\\[" y "\\]" significan que lo que hay entre ellos son caracteres no imprimibles, o no visualizables (colores en este caso) es decir, que se verá el color, por ejemplo, pero no la secuencia de caracteres que lo define.

El resto de los caracteres se visualizan tal cual. Algunos caracteres están precedidos de dos '\\'. La primera barra es necesaria para que la segunda tenga efecto. Si para obtener el nº del terminal hay que poner "\\#", y no anteponeamos otra '\\' será como si bash "no viera" la barra que precede al carácter '#'. Es decir: que "\\\" bash "lo ve" como '\\'. La barra inversa '\\', que se denomina el "carácter escape", sirve como una especie de marca para reconocer otros caracteres especiales, como el retorno de carro o carácter de nueva línea (\\n), por ejemplo.

```
PS2="$COLOR->\\[\\033[0m\\] "
```

Esto es el prompt PS2, que aparecerá cuando queramos continuar un comando en la siguiente línea, para lo que hay que escribir un carácter '\\ y pulsar Intro. El segundo prompt utilizará el mismo color base que el primero y los caracteres "-> ". A partir de ahí, lo que sigue es lo mismo que la última parte del prompt PS1.

## 5. Poniéndolo a funcionar

Una vez que tenemos la función lista para realizar el cambio del prompt, sólo hay que llamarla desde el lugar apropiado. Lo siguiente que hay que hacer es situarla en algún directorio accesible para todos los usuarios y darle permiso de lectura, el de ejecución no es necesario en este caso. Un buen lugar para esto es el directorio `/usr/local/bin`. Para quienes no sepan mucho sobre la organización de los archivos en un sistema Linux, y se hagan preguntas acerca de dónde puñetas están las cosas, trataré de resumir lo que he aprendido al respecto:

El directorio raíz se llama simplemente `'/'`. A partir de él cuelga todo el sistema de archivos de un Linux. No suele contener archivos propiamente dichos, si no algunos enlaces al kernel del sistema y al `System.map` para que el cargador de arranque los encuentre ahí, pero esto no es obligatorio. De hecho, en mi sistema Debian GNU/Linux no hay un sólo archivo ni enlace simbólico en el `/`, ya que tengo a Grub configurado para que vaya a buscar lo que necesita directamente a `/boot`. Ya que hemos mencionado a `/boot`, diremos que es ahí donde se esconde el kernel y los archivos necesarios para su puesta en marcha. También suele haber un archivo, llamado `config-[versión-kernel]`, que guarda la configuración de su kernel, y que es posible utilizar para recompilar uno nuevo quitándole lo que nos sobre y/o añadiéndole lo que nos falte, sin necesidad de recorrer todas las numerosas opciones de configuración que un kernel Linux tiene. Luego hay una serie de directorios, o enlaces, para el caso da lo mismo, que se suelen llamar `/media`, `/floppy`, `/cdrom`... etc. Es ahí donde se montan las unidades extraíbles de disquetes, cdrrones, dvds y demás. En `/bin` están los programas pertenecientes al sistema (sí, eso mismo, los comandos) disponibles para todos los usuarios, mientras que en `/sbin` están los comandos que sólo puede usar el root. En `/lib` se encuentran una serie de bibliotecas de código necesarias para el sistema y, en `/lib/modules/[version-kernel]` se esconden los módulos disponibles para el kernel, caso de haberlos, ya que es posible compilar un kernel que no utilice módulos. En la jerga linuxera, un módulo viene a ser un driver, para entendernos. En `/dev` están los dispositivos, y hay una cantidad enorme, como cualquiera podrá comprobar. El primer disco duro es `/dev/hda`, la unidad de disquetes es `/dev/fd0`, el CD-ROM suele ser `/dev/hdc`, el primer puerto serie es `/dev/ttyS0`, la impresora en el puerto paralelo, `/dev/lp0`. Si hiciéramos algo como, por ejemplo: `ls /bin > /dev/lp0`, estaríamos enviando la salida del comando directamente a la impresora, tal cual y a lo bruto. Pero sigamos con los directorios: `/root` es la casa del root, obviamente. Cuando root se conecta directamente, o mediante `su -`, va a parar a `/root`. Los directorios `/sys` y `/proc` quedan reservados a "cosas del sistema", digamos que el hecho que `/sys` exista puede hacer disminuir el número de dispositivos de `/dev`, y que `/proc` es un directorio virtual con información obtenida directamente del sistema en ejecución. `/tmp` es para uso temporal, en `/var` estarían los archivos correspondientes a las colas de impresión, las tablas de las bases de datos, los logs del sistema (`/var/log`) y, en fin, archivos que pueden considerarse de contenido muy variable. En `/etc` están los archivos de configuración genérica de los distintos programas que tengamos instalados. Ahí está `/etc/fstab`, con las opciones de montaje, `/etc/init.d`, con los scripts de los servicios que utilicemos... etc, etc, :-). En `/home` están los directorios pertenecientes a los usuarios normales, y en `/usr` hay un mogollón de cosas: `/usr/bin`, los programas que tenemos instalados, `/usr/lib` las bibliotecas necesarias para esos programas, `/usr/include` los archivos de las versiones de desarrollo de los programas. `/usr/local` suele contener los programas instalados por nosotros mismos desde los fuentes, sin utilizar el sistema de instalación de la distribución. A `/usr` va a parar casi todo lo que nos instalamos aparte del sistema propiamente dicho, juegos incluidos (`/usr/games`). `/usr/share/doc` es siempre un lugar interesante para los curiosos :-). Hay otros archivos dentro del directorio `$HOME` de cada usuario, que se denominan "ocultos" y que comienzan por un punto. En esos archivos y directorios se guardan las configuraciones y

preferencias de cada usuario para cada programa que lo requiera. Pueden listarse utilizando los comandos **ls -a** o **ls -la**. Por ejemplo, un ordenador que usan tres personas y todas utilizan también el programa de correo mozilla-thunderbird, puede tener en el directorio `.mozilla-thunderbird` de cada usuario sus propias cuentas y sus propios mensajes, sin que cada uno sepa nada de lo que tienen los otros. O cada uno puede tener la configuración de su gestor de ventanas preferido, gnome, kde, xfce, wmaker... etc. Estas cosillas forman parte de lo que hace a un sistema multiusuario. Bueno, hasta aquí más o menos lo que sé, o creo saber, sobre estas materias.

Una vez hecho el inciso anterior, continúemos instalando nuestro prompt. Decíamos que un buen lugar para situarlo sería `/usr/local/bin`. Pues bien, lo copiamos allí con **cp super\_prompt /usr/local/bin**, para lo que tenemos que estar conectados como root, porque los usuarios normales no suelen tener permisos de escritura fuera de su \$HOME y de /tmp. Después, y por si acaso, le damos los permisos necesarios: **chmod 0644 /usr/local/bin/super\_prompt**

Lo siguiente a realizar será editar dos archivos del directorio \$HOME de cada usuario que quiera utilizar el prompt modificado: `.bashrc` y `.bash_profile`.

En `.bashrc` habrá que poner algo así:

```
if [ -f /usr/local/bin/super_prompt ]; then
source /usr/local/bin/super_prompt
super_prompt
else
# prompt por defecto:
PS1='\u@\h:\w\$ '
fi
```

Esto viene a significar:

```
if [ -f nombreArchivo ]; then
    Si existe nombreArchivo, entonces

source nombreArchivo
```

Incluye el archivo fuente en ese punto y, para que se ejecute el código que contiene, sólo hay que llamarlo por su nombre, como figura en la línea de abajo. Si en lugar de "-f" pusiéramos "-d", estaríamos comprobando la existencia de un directorio.

Aquí hay que tener en cuenta que el prompt por defecto no quede fuera de lugar, por si acaso fallase el nuevo, colocándole un else, como en el ejemplo. Y, por último, sólo habrá que comprobar que en `.bash_profile` exista una línea como esta: (Si no existe se añade, y si estuviese comentada con un '#' al principio, se descomenta).

```
if [ -e ~/.bashrc ]; then
. ~/.bashrc
fi
```

Aquí, "-e" también significa "si el fichero existe". La diferencia con "-f", según las páginas man, es que -f indica que el fichero debe ser regular, lo que interpreto como que no podrá ser un enlace simbólico, un fichero de dispositivo, y cosas así. El punto antes de ~/bahsrc se utiliza para que el archivo se ejecute directamente.

## 6. Algunas cosillas más, y terminamos

Hay cosas que en emacs pueden no ser evidentes. Una de ellas es, por ejemplo, el tabulador. ¿Qué diablos hay que hacer para que emacs inserte una tabulación? Pues Control-q, liberar Control y pulsar el tabulador (C-q-tab). Fácil, ¿no? Pero eso es porque el tabulador en emacs se ocupa de indentar correctamente las líneas de código y, en general, cualquier línea para la que tenga un modo, como sería el caso del XML/SGML.

¿Qué significa ~/? Pues nuestro directorio \$HOME. Si nuestro nombre de usuario es pepe, ~/ y \$HOME hacen referencia al mismo sitio: /home/pepe.

¿Qué significa "montar" algo en linux? Pues, así de golpe, la respuesta sería: "hacer accesible un dispositivo", pero a ver si puedo aclararme mejor. Posiblemente tu distribución esté configurada con algo llamado **automount**, y cuando insertas un disquete o un CD inmediatamente lo pone a tu disposición en el escritorio, con un bonito iconillo. Pero esto no tiene por qué ser así, y si no prueba en una Debian "a pelo" y verás. Cuando no hay configurado ningún sistema de automontaje, hay que montar a mano de la siguiente forma: **mount /dispositivo -t sistema-de-archivos /directorio** Para un disquete con formato windows fat-32 (antes del NTFS) sería: **mount /dev/fd0 -t vfat /floppy** lo que significa: montar el dispositivo llamado /dev/fd0 (la primera unidad de disquetes) cuyo sistema de archivos es de tipo vfat (fat32, el del Windows) en el directorio llamado /floppy (el directorio llamado /floppy tiene que existir). A partir de ese momento, para ver el contenido del disquete sólo habrá que mirar dentro del directorio /floppy (ls /floppy). Pero cuidado, las unidades extraíbles deben ser desmontadas antes de extraerlas, para lo cual, en este caso, será suficiente el comando: **umount /floppy**. Esto mismo se aplica a cada unidad a la que deseemos tener acceso, ya sean particiones de la máquina local o unidades de otras máquinas accesibles en una red.

## 7. Enlaces relacionados

*bash.* (<http://es.wikipedia.org/wiki/Bash>)

*Programación en bash.*

(<http://es.tldp.org/COMO-INSFLUG/COMOs/Bash-Prog-Intro-COMO/Bash-Prog-Intro-COMO.html>)

*Un "cómo" sobre el prompt en bash.*

(<http://es.tldp.org/COMO-INSFLUG/COMOs/Bash-Prompt-Como/>)

*Ampliando explicaciones sobre el prompt que a mí me gusta*

*Algunas cuestiones básicas sobre software libre, programación y Linux.*  
(<http://www.iskiamjara.com/aprendeinformatica/contentid-14.html>)