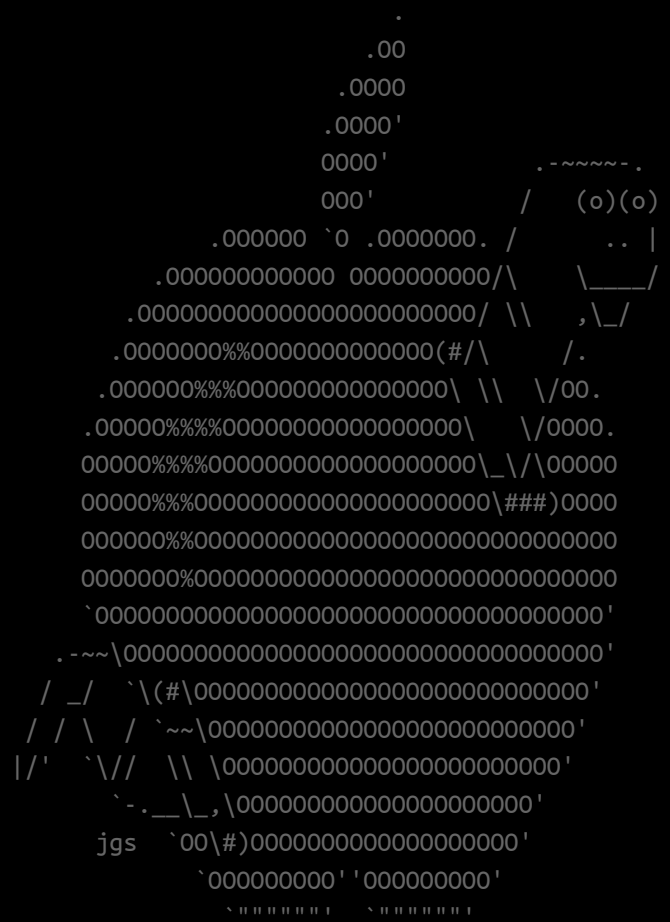
Año 0 - Número 12
28.10.2013

Hackers & DEVELOPERS Change

HACKER TEAM:
María José Montes Díaz
Pablo Bernardo Baragaño
Milagros Infante Montero
y colaboradores



Hacker

es alguien que disfruta

jugando con la inteligencia

HACKERS & DEVELOPERS MAGAZINE
MAGAZINE DIGITAL DE EDICIÓN MENSUAL SOBRE SOFTWARE LIBRE, HACKING Y PROGRAMACIÓN

Hackers & Developers Magazine
Creado por **Eugenia Bahit**

HACKERS & DEVELOPERS MAGAZINE "CHANGE" - AÑO 0 NÚMERO 12
FECHA DE PUBLICACIÓN: 28 DE OCTUBRE DE 2013

CREADO CON **LIBREOFFICE WRITER**

CHANGE

HACKERS & DEVELOPERS MAGAZINE

NUESTRO COMPROMISO CON QUIENES APUESTAN AL VERDADERO CAMBIO

PAPERS DE • MARÍA JOSÉ MONTES DÍAZ • MILAGROS INFANTE MONTERO • PABLO BERNARDO • Y COLABORADORES

EN ESTA EDICIÓN:

Explorando lenguajes de programación con repl.it.....3
Cifrando nuestros datos en Archlinux: EncFS.....9
Constructores en Javascript.....16
VIM y el movimiento eficiente.....20
Sudoers: Cambiando de identidad.....24
La terminal en GNU/Linux: "Back to the future?".....27
Persona: El login libre de Mozilla.....32
Licencias del Software Libre y Libertades del Usuario.....36

Explorando lenguajes de programación con repl.it

Cuando estamos programando, necesitamos compilar nuestro código y hacer las respectivas pruebas para saber si tuvimos errores, si hay algo por refactorizar o si lo hicimos bien. Repl.it es una herramienta en línea que nos permitirá compilar el código de diversos lenguajes de programación.

Escrito por: **Milagros Alessandra Infante Montero** (Est. Ing. Informática)

Estudiante de Ingeniería Informática. Miembro de la comunidad de software libre **Lumenhack**. Miembro del equipo de traducción al español de **GNOME**. Apasionada por el desarrollo de software, tecnología y gadgets. Defensora de tecnologías basadas en software libre y de código abierto.

Webs:
Blog: www.milale.net

Redes sociales:
Twitter: **@milale**

Repl.it[1] es un entorno en línea que nos permitirá explorar los lenguajes de programación de manera interactiva, el nombre proviene de read-eval-print loop, que es a su vez un entorno de programación. En un REPL el usuario ingresa una o más expresiones, que son evaluadas y finalmente se muestran los resultados.

El nombre viene en mención a las antiguas funciones de Lisp que implementan la funcionalidad:

- **read:** función que acepta una expresión de parte del usuario.
- **eval:** función que toma la estructura de datos interna y la evalúa.
- **print:** función que toma el resultado dado por *eval* y lo imprime al usuario.

REPL facilita la programación exploratoria y la depuración; porque el bucle read-eval-print es usualmente más rápido que el clásico ciclo editar-compilar-ejecutar-depurar.

1 http://repl.it/languages

Empezando a usar repl.it

Esta herramienta repl.it es muy sencilla de usar, debemos seleccionar el lenguaje de programación en el que estemos desarrollando, luego en la consola del lado derecho se debe escribir una expresión y finalmente presionar "Enter" para ver los resultados impresos en la consola de la izquierda. Incluso, repl.it interpreta y soporta variables, estados y efectos en los diversos lenguajes que soporta.

Repl.it provee un emulador de terminal con muchas características y además un editor de código para actualmente 17 lenguajes de programación: Qbasic (programación estructurada para principiantes), Forth (interactivo lenguaje orientado a pila), Emoticon (programando con una dosis extra de sonrisas), Brainfuck (un controlador de máquina de Turing puro), LOLCODE (El lenguaje básico de lolcats), Unlambda (Pureza funcional hecha forma), Bloop (bucles delimitados), Ruby (un lenguaje dinámico y natural orientado a objetos), Python (lenguaje dinámico que da énfasis a la legibilidad del código), Lua (lenguaje liviano de script multi-paradigma), Scheme (Un dialecto dinámico elegante de Lisp), JavaScript (el lenguaje de facto de la web), JavaScript.next (el JavaScript del mañana), Move (la manera fácil de programar en la web), Kaffeine (JavaScript avanzado para profesionales), CoffeeScript (JavaScript no fantástico) y Roy (Lenguaje funcional pequeño que compila a JavaScript).

Características de repl.it

Repl.it permite guardar la sesión en la que estás trabajando e incluso compartirla con otros usuarios mediante una URL o en redes sociales. Si necesitas cerrar la sesión, al volver, puedes ejecutarla nuevamente con el enlace creado y continuar donde te quedaste previamente.

> Repl.it es open source, el código puede ser explorado e incluso puedes contribuir en varias partes a través de los repositorios de Github https://github.com/replit[2].

Esta herramienta puede ser usada en cualquier dispositivo que tenga un navegador web moderno, ya que actualmente cuentan con soporte total para dispositivos Android™ y soporte parcial para iOS. Todo código procesado por repl.it se ejecuta por completo en la computadora, usa intérpretes de lenguajes escritos en JavaScript para ejecutar código y mantener el seguimiento del estado. A continuación listaré los atajos de teclado más usados en la consola:

Enter	Evalúa el comando ingresado.
Shift + Enter	Continúa a la siguiente línea.
Tab	Indentar.
Shift + tab	Sacar indentado.
Arriba	Elemento de la historia anterior.
Abajo	Elemento de la historia siguiente.
Ctrl + arriba	Mover a la línea arriba del cursor.

2 https://github.com/replit

Ctrl + abajo	Mover a la línea debajo del cursor.
Ctrl + E	Mover al final de la línea actual.
Ctrl + A	Mover al principio de la línea actual.
Ctrl + Z	Cancelar la terminal actual.
Ctrl + L	Abrir el selector de lenguaje.
Ctrl + G	Abrir la vista previa de ejemplos.
Ctrl + H	Abrir la página de ayuda.
Ctrl + S	Guardar la sesión (nueva revisión).
Escape	Cerrar vista actual.

Pruebas con los lenguajes de programación

Repl.it divide los lenguajes de programacion que soporta en cuatro “categorías”: Lenguajes clásicos, esotéricos, prácticos y web. A continuación veremos unos ejemplos de cómo podemos usar repl.it al desarrollar.

- **QBasic:**

  Un ejemplo sencillo con QBasic podría ser de la siguiente manera:

```
PRINT "Hola lector de H&D. Porfavor escribe tu nombre: "
INPUT NOMBRE
PRINT "Bienvenido a la edición número 12, "; NOMBRE; "!"
```

  Y al compilarlo, obtendremos esto como resultado:

```
QBasic (qb.js)
Copyright (c) 2010 Steve Hanov
Hola lector de H&D. Porfavor escribe tu nombre:
Milale
Bienvenido a la edición número 12, Milale!
```

  Aquí esta el enlace del ejemplo para que puedan jugar con el: http://repl.it/Lce

- **Emoticon:**

  Emoticon nos permite escribir código con una recarga de sonrisas, así:

```
:-O Sonrian lectores :-Q S:-P :-Q
```

  Obteniendo como resultado:

```
Emoticon v1.5 (emoticoffee)
Copyright (c) 2011 Amjad Masad

Sonrian lectores=>
X: 7
Z: START,:-O,Sonrian,lectores,:-Q,S:-P,:-Q
A: :
G:
S:
E:
::
```

  De la misma manera, aquí tienen el enlace al ejemplo: http://repl.it/Lcf

- **Brainfuck:**

  Brainfuck es un lenguaje de programación que hace pensar mucho al desarrollador, veamos un ejemplo:

```
[Este programa imprime el triángulo de Sierpinski.]
                              >
                             + +
                            +   +
                           [ < + +
                          +       +
                         + +     + +
                        >   -   ]   >
                       + + + + + + + +
                      [               >
                     + +             + +
                    <   -           ]   >
                   > + + >         > > + >
                  >       >       +       <
                 < <     < <     < <     < <
                <   [   -   [   -   >   +   <
               ] > [ - < + > > > . < < ] > > >
              [                               [
             - >                             + +
            +   +                           +   +
           + + [ >                         + + + +
          <       -                       ]       >
         . <     < [                     - >     + <
        ]   +   >   [                   -   >   +   +
       + + + + + + + +                 < < + > ] > . [
      -               ]               >               ]
     ] +             < <             < [             - [
    -   >           +   <           ]   +           >   [
   - < + >         > > - [         - > + <         ] + + >
  [       -       <       -       >       ]       <       <
 < ]     < <     < <     ] +     + +     + +     + +     + +
+   .   +   +   +   .   [   -   ]   <   ]   +   +   +   +   +
* * * * * M a d e * B y : * N Y Y R I K K I * 2 0 0 2 * * * * *
```

  Y el resultado obtenido pueden verlo aquí al ejecutarlo: http://repl.it/Lcg

- **LOLCODE:**

  LOLCODE es un lenguaje de programación muy divertido por las palabras que usa para su sintaxis, como veremos a continuación:

```
HAI
VISIBLE "Hola lectores de Hackers & Developers Magazine!"
KTHXBYE
```

  Y como resultado obtendremos:

```
LOLCODE v1.2 (lol-coffee)
Copyright (c) 2011 Max Shawabkeh

Hola lectores de Hackers & Developers Magazine!
```

  Y si quieren jugar con este lenguaje aquí está el enlace del ejemplo: http://repl.it/Lch

- **Unlambda:**

  Unlambda también tiene una sintaxis distinta como vemos a continuación:

```
   `r```````````.L.e.c.t.o.r.e.s. .H.Di
```

  Obtenemos como resultado esto:

```
Unlambda v2.0 (unlambda-coffee)
Copyright (c) 2011 Max Shawabkeh

Lectores HD
=> i
```

  Aquí está el enlace del ejemplo: http://repl.it/Lcj

- **Ruby:**

  Un ejemplo de desarrollo en Ruby sería así:

```
   puts 'Hola, ingresa una frase para los otros lectores:'
   puts 'Lectores, ' + gets.strip + '!'
```

  Y el resultado obtenido sería el siguiente:

```
Unlambda v2.0 (unlambda-coffee)
Copyright (c) 2011 Max Shawabkeh

Lectores HD
=> y the force be with you
Lectores, May the force be with you!
=> nil
```

  El enlace del ejemplo está aquí: http://repl.it/Lck

- **Python:**

  Nuestro querido lenguaje de programación también puede ser compilado con repl.it, aquí tenemos un ejemplo:

```
###Pares e impares
numero = input('Introduzca un numero: ')
if numero%2 == 0:
    print 'Este numero es par'
else:
    print 'Este numero es impar'
```

  Y obtenemos lo siguiente como resultado:

```
Python 2.7.2 (default, Jul 20 2011, 02:32:18)
[GCC 4.2.1 (LLVM, Emscripten 1.5, Empythoned)] on linux2

Introduzca un numero:  14
Este numero es par
```

  El enlace para el ejemplo esta aquí: http://repl.it/Lcm

- **Kaffeine:**

  El lenguaje JavaScript para profesionales también está aquí, tenemos un ejemplo:

```
  console.log "Cuál es tu nombre HD reader?"
  console.log "No te pierdas la edición 12 de HD, #{console.read!}!"
```

  Y obtenemos lo siguiente como resultado:

```
| |/ /__ _ / _|/ _|___(_)_ _  ___
| ' </ _` |  _|  _/ -_) | ' \/ -_)
|_|\_\__,_|_| |_| \___|_|_||_\___|
Version 0.0.4, Copyright (c) 2010 Jonah Fox

Cuál es tu nombre HD reader?
 Roy
No te pierdas la edición 12 de HD, Roy!
```

  Y el enlace del ejemplo lo tenemos aquí: http://repl.it/Lcn

- **CoffeeScript:**

  Con CoffeeScript también podemos hacer un ejemplo:

```
  console.log 'Hola lectores, miren los numeros al cuadrado desde el 10 al 1'
  cuadrados = (x) -> x * x
  console.dir (cuadrados numero for numero in [10..1])
```

  Obteniendo lo siguiente como resultado:

```
CoffeeScript v1.3.1
Copyright (c) 2011, Jeremy Ashkenas

Hola lectores, miren los numeros al cuadrado desde el 10 al 1
[ 100, 81, 64, 49, 36, 25, 16, 9, 4, 1 ]
```

  Este es el enlace del ejemplo: http://repl.it/Lco

- **Roy:**

  Este lenguaje de programación aunque poco conocido es un lenguaje funcional pequeño que compila a JavaScript, aquí veremos un ejemplo:

```
  log "Ingresa tu edición de HD favorita: "
  console.read (\n -> console.log "No dejes de disfrutar " ++ n ++ " en hdmagazine.org")
```

  El resultado que obtendremos es el siguiente:

```
Roy 0.1.3
Copyright (C) 2011 Brian McKenna

Ingresa tu edición de HD favorita:
 Champagne
No dejes de disfrutar Champagne en hdmagazine.org
```

  Aquí está el enlace al ejemplo: http://repl.it/Lcp

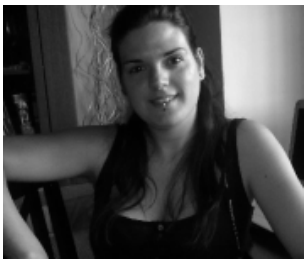
Existen muchos lenguajes de programación de los que no se conoce mucho pero porque no empezar de a pocos a aprenderlos, quizás presentan características que pueden gustarnos e incluso llegar a convertirse en nuestro lenguaje favorito.

Como hemos podido apreciar y comprobar, repl.it es una herramienta en línea muy poderosa que nos permite probar nuestro código total o por partes; y así poder saber que hicimos bien o mal, contar con una herramienta de este tipo es realmente fundamental al momento de “codear”, lo más importante es continuar con un **Happy Hacking!!!**

Cifrando nuestros datos en Archlinux: EncFS

Algo que debemos tener en cuenta es cifrar nuestros datos, al menos aquellos que no queramos que sean públicos. En caso de robo o extravío de nuestro equipo, podría accederse al disco montándolo en otro o arrancando con otro sistema. Veamos cómo proteger esos datos.

Escrito por: **María José Montes Díaz** (Docente & Programadora)



Estudiante de Grado Ingeniería en Tecnología de la información. Técnico en informática de gestión. Monitora FPO. Docente de programación Python y Scratch para niños de 6-12 años. Activista del software libre y cultura libre.

Webs:
Blog: http://archninfa.blogspot.com.es/

Redes sociales:
Twitter: **@MMontesDiaz**

En este artículo voy a centrarme en proteger los datos del usuario. Veremos una herramienta que nos permitirá probar el cifrado de datos en nuestro sistema.

La herramienta que vamos a utilizar es a nivel de cifrado de archivo. Existen dos maneras, básicamente, de cifrar nuestros datos: A nivel de disco y a nivel de archivos.

El primero, protegerá el sistema siempre que esté apagado. En el arranque, el disco se montará y, una vez iniciado el sistema operativo, no estará cifrado. El segundo, protege archivos y directorios. Tiene la ventaja de poder utilizarlo de forma selectiva y, aunque el sistema esté arrancado, no tiene por qué estar descrifado.

Con el cifrado a nivel de archivo, podemos hacer copias de seguridad sin necesidad de descifrar el contenido. Esto es útil en muchos casos. Como desventaja, dejamos ver la cantidad de archivos que hay en un directorio, estructura y tamaño de archivos, por ejemplo.

Para una protección robusta, sería interesante utilizar ambos. De hecho, en distribuciones como Debian, Ubuntu y Fedora, permiten utilizar un cifrado de disco, con **dm-crypt**, en el proceso de instalación. En Ubuntu, además, se puede cifrar el **home** de los usuarios, utilizando **eCryptfs**.

En este artículo me voy a centrar en el cifrado individual de archivos/directorios.

¿Por qué EncFS?

Es una herramienta muy fácil de utilizar y no necesitamos privilegios de root para poder usarla. Nos permite, en el espacio de usuario, realizar los cifrados y, mediante **FUSE**, montar las carpetas para poder acceder a su contenido. Está disponible en la mayor parte de distribuciones GNU/Linux y su utilización es muy similar, por no decir igual.

Un punto a su favor es la portabilidad. Está disponible para diferentes plataformas, incluidas privativas, lo que nos permite gran flexibilidad. Además, no es necesario recordar los parámetros con los que se configuró el directorio cifrado.

Sin embargo, **eCryptfs** forma parte del núcleo Linux, lo que, según sus autores, permite que sea más rápida. Esto, en teoría, debe ser así, pues no hay que hacer cambio de contexto entre el kernel y el espacio de usuario.

Por otra parte, las herramientas y scripts para utilizar **eCryptfs** se desarrollaron pensando en Ubuntu, con lo que en otros sistemas, puede que no nos funcionen, al menos que utilicemos una configuración exactamente igual a la realizada por esta distribución.

Cifrando nuestros datos

Vamos a instalar **encfs** en nuestra Archlinux:

```
# pacman -Sy encfs
```

Ahora vamos a crear el directorio cifrado e indicaremos dónde vamos a montarlo descifrado:

```
$ encfs ~/.prueba ~/prueba
```

Un detalle a tener en cuenta es que debemos utilizar las rutas completas. Además, su utilización es con nuestro usuario. Nos pedirá el modo de cifrado y la contraseña a utilizar:

```
[mary@archninfa ~]$ encfs ~/.prueba ~/prueba
El directorio "/home/zodiac/.prueba/" no existe. ¿Deberia ser creado? (s,n) y
El directorio "/home/zodiac/prueba" no existe. ¿Deberia ser creado? (s,n) y
Creando nuevo volumen cifrado.
Por favor, elige una de las siguientes opciones:
 pulsa "x" para modo experto de configuracion,
 pulsa "p" para modo paranoia pre-configurado,
 cualquier otra, o una linea vacia elegira el modo estandar.
?> p
```

```
Seleccionada configuración Paranoica.

Nueva contraseña Encfs:
Verifique la contraseña Encfs:
```

El método estándar es muy seguro y utiliza por defecto los siguientes valores:

```
            Cipher: AES
            Key Size: 192 bits
            PBKDF2 with 1/2 second runtime, 160 bit salt
            Filesystem Block Size: 1024 bytes
            Filename Encoding: Block encoding with IV chaining
            Unique initialization vector file headers
```

El método paranoia, el que he utilizado, es todavía más seguro, pero impide la utilización de “enlaces duros”, lo que puede hacer que algunas aplicaciones fallen, como es el caso de **mutt**. Sus valores son:

```
            Cipher: AES
            Key Size: 256 bits
            PBKDF2 with 3 second runtime, 160 bit salt
            Filesystem Block Size: 1024 bytes
            Filename Encoding: Block encoding with IV chaining
            Unique initialization vector file headers
            Message Authentication Code block headers
            External IV Chaining
```

Ya hemos creado el directorio cifrado y el punto de montaje, además está montado. Para utilizarlo, bastará copiar aquellos archivos que queramos que estén cifrados a **~/prueba**. Para desmontar el directorio, tan solo debemos ejecutar:

```
$ fusermount -u ~/prueba
```

Algunos detalles a tener en cuenta:

- El directorio donde vamos a montar debe estar vacío.
- Los archivos que copiemos directamente al directorio donde están alojados los archivos cifrados, no serán visibles en el punto de montaje.

Podemos cambiar la clave inicial con el comando:

```
$ encfsctl passwd ~/.prueba
```

Como no todo tiene porqué ser línea de comandos, disponemos de varias herramientas que nos permiten hacer todo esto en nuestro entorno gráfico:

- **CryptKeeper**: Es un icono bara la bandeja de sistema, que nos permite crear, modificar y montar/desmontar directorios cifrados, interesante para entornos GTK.

- **kde-servicemenus-encfs**: Es un servicio para Dolphin/Konqueror, que nos permite crear y montar/desmontar directorios cifrados, desde el menú del navegador de archivos.

- **Kencfs**: Es una aplicación, similar a las anteriores. También tiene su icono para la bandeja de sistema.

Todo esto es para un montaje manual. Estas herramientas se encuentra en el AUR, así que podemos utilizar yaourt para instalarlas (o cualquier otra herramienta que nos permita acceder al AUR) o descargarlas y crear el paquete.

Si queremos que los directorios cifrados se monten automáticamente al iniciar la sesión, tendremos que utilizar **pam_encfs**. Antes de nada, instalemos:

```
yaourt -Sy pam_encfs
```

O bien:

```
wget https://aur.archlinux.org/packages/pa/pam_encfs/pam_encfs.tar.gz
tar -xvzf pam_encfs.tar.gz
cd pam_encfs
makepkg -si
```

Ahora vamos a configurar. Para ello editamos el archivo /etc/security/pam_encfs.conf, comentamos la línea, añadiendo una # al principio:

```
#encfs_default --idle=1
```

Esta opción desmontará el directorio tras un minuto en desuso. Si vamos a cifrar nuestro **home**, no nos interesa. Aunque, junto con la opcion **-- ondemand**, que montará el directorio bajo demanda, junto con **--extpass=/ruta/programa**, que ejecutará "programa" para solicitar la clave, puede ser interesante.

Al final del archivo, añadimos nuestro directorio. En este caso, para continuar con el ejemplo:

```
#USERNAME SOURCE                    TARGET PATH           ENCFS Options  FUSE Options
mary      /home/mary/.prueba        /home/mary/prueba     -v              allow_other
```

Indicamos el usuario, el directorio cifrado, dónde lo vamos a montar, modo verbose (para ver la salida) y las opciones FUSE (**allow_other** permite el acceso al punto de montaje a todos usuarios, según los permisos del sistema de archivos; **allow_root** solo permite acceso al usuario que montó y al root).

Detalles a tener en cuenta:

- Debemos especificar al menos una opción en **ENCFS Options**. Si no queremos detalles, podemos cambiar **-v** por **--standar**.

- Para poder utilizar **allow_user**, en el archivo /etc/fuse.conf hay que descomentar la línea **user_allow_other** y, en el /etc/security/pam_encfs.conf, comprobar qué opción hay en **fuse_default**, pues si aparece **allow_root**, nos dará un error, porque intentará montar con las dos opciones y, éstas, son excluyentes.

- La clave de usuario y de cifrado deben ser iguales.

Una vez hecho esto, hay que configurar algunas cosas más: Si vamos a utilizar un gestor de inicio de sesión gráfico (**KDM** o **GDM**), tan solo hay que editar el archivo /etc/pam.d/system-auth y añadir al principio del archivo:

```
auth      sufficient pam_encfs.so
```

Además, revisamos que la línea **auth required pam_unix.so** está así:

```
auth      required   pam_unix.so     try_first_pass nullok
```

Al final del archivo añadimos:

```
session   required  pam_encfs.so
```

Dado que algunos gestores de inicio de sesión, como **KDM** y **GDM**, además del **SSH**, utilizan la configuración de /etc/pam.d/system-auth, por lo que será suficiente para el montaje automático. Si utilizamos alguno que no la utilice, deberemos incluir esas líneas en su archivo correspondiente. Para el caso de **SLIM** es /etc/pam.d/slim.

Para que se monte al iniciar sesión en un terminal, dado que no utiliza la configuración de /etc/pam.d/system-login, que a su vez incluye la configurar de /etc/pam.d/system-auth, editamos el archivo /etc/pam.d/login, y añadimos la línea **auth sufficient pam_encfs.so**, además de revisar la línea **auth required pam_unix.so**:

```
. . .

auth            required        pam_securetty.so
auth            requisite       pam_nologin.so
auth            sufficient      pam_encfs.so
auth            required        pam_unix.so nullok try_first_pass

. . .

#Si vamos a iniciar sesión con login gráfico, NO añadir la línea siguiente
#al cerrar sesión en la terminal, intentará cerrar aunque tengamos alguna
#sesión gráfica abierta
#session         required        pam_encfs.so
```

Ya tenemos todo listo, podemos comprobar que el montaje funciona ingresando en el sistema desde un terminal (ALT+CTRL+F2, por ejemplo, para ir a una). Antes de iniciar la sesión deberemos haber desmontado el directorio.

Cifrando un HOME existente

Ahora vamos a cifrar un **HOME** que ya exista. Prepararemos una estructura de directorios para alojar los datos cifrados de los usuarios. Primero crearemos el directorio que alojará los contenedores:

```
sudo mkdir /home/.private
sudo chown og-rw /home/.private
```

Le quitamos los permisos de lectura/escritura, pero no el ejecución: No se podrá listar el contenido del directorio, pero podremos acceder a nuestro contenedor.

Creamos el contenedor:

```
encfs ~/nombre_usuario ~/temporal
fusermount -u ~/temporal
sudo mv ~/nombre_usuario /home/.private
rm -r ~/temporal
```

Ahora hacemos un backup de los datos, creamos el punto de montaje y restauramos:

```
cd /home
sudo mv nombre_usuario nombre_usuario.backup
sudo mkdir nombre_usuario
sudo chown nombre_usuario:grupo_usuario nombre_usuario
encfs /home/.private/nombre_usuario /home/nombre_usuario
sudo mv nombre_usuario.backup/* nombre_usuario.backup/.* -t nombre_usuario
```

Una vez hecho esto, vamos a añadir la siguiente línea al /etc/security/pam_encfs.conf:

```
#USERNAME       SOURCE                  TARGET       ENCS Options      FUSE Options
-               /home/.private           -           -v                allow_other
```

El **-** se sustituye por nombre de usuario que inicia sesión. Al ponerlo, el campo **SOURCE** se sustituye por ruta+usuario. Si ponemos *, la ruta seria **~/+usuario**. En el campo **TARGET**, si es **-** sustituye por la ruta del **HOME** del usuario.

Referencia:
https://wiki.archlinux.org/index.php/EncFS

Constructores en Javascript

Para comprender lo importante de Javascript en el mundo actual de la programación, debemos conocer como se implementan los patrones con este genial lenguaje, en este artículo veremos a que le llaman patrón constructor en Javascript.

Escrito por: **Sergio Infante Montero** (Ingeniero de Software)

Ingeniero Informático con estudios de Master de Dirección Estratégica en TI. Senior Software Developer en **Belatrix,** activista, contribuidor y consultor de proyectos **FLOSS.** Miembro de **GNOME Foundation.** Escritor de artículos y libros técnicos de programación.

Webs:
http://about.me/neosergio

Redes sociales:
Twitter:**@neosergio**

Esta demás mencionar que para ser un programador profesional, se deben conocer patrones[3], la implementación de estos varía de acuerdo al lenguaje de programación, las diferencias radican en la sintaxis propia de cada tecnología y las limitaciones de la misma, sin embargo, en general tienen una misma manera de trabajar y funcionar, de hecho por ello se llaman patrones.

El llamado patrón constructor, en la programación orientada a objetos clásica, en realidad es un método especial que se usa para iniciar un objeto recientemente creado (se crea una vez que se ha separado un espacio en memoria para el).

Para Javascript esto es diferente y ahí radica la confusión con llamarlo patron constructor o simplemente método, en Javascript casi todo es un objeto, así que en particular esta manera de crear objetos al cual se le llama patrón (aunque en terminos conceptuales, no lo es) se refiere a los objetos constructores. Los objetos constructores son usados para crear tipos específicos de objetos, preparados para aceptar argumentos, métodos y propiedades.

Creación de objetos

Para la creación de objetos, existen tres formas comunes en Javascript.

3 En las primeras ediciones de esta revista, hay un artículo que habla con mayor detalle sobre los patrones.

```
//Estas son las tres formas de crear un objeto en Javascript
var nuevoObjeto = {};

var nuevoObjeto = Object.create(null);

var nuevoObjeto = new Object();
//En todos los ejemplos anteriores se crean objetos vacios
```

Para probar fácilmente el código se puede usar interpretes como rhino, jsc o node, que pueden ser convocados desde una terminal. Esto funciona para Microsoft Windows, GNU/Linux o Mac OS X

La documentación de Object puede ser revisada en la documentación de Mozilla Developer Network[4], incluso si se desea se puede contribuir a su traducción a otros idiomas.

Ya que los objetos han sido creados sin contenido, existen 4 formas comunes de asignar valores y llaves a un objeto, las dos simples son las siguientes:

```
//La primera forma es con la sintaxis del punto

//Asignar propiedades. El SET
nuevoObjeto.algunaLlave = "HD Magazine";
//Obtener propiedades. El GET
var valor = nuevoObjeto.algunaLlave;

//La segunda forma la sintaxis del corchete, muy similar a la anterior
//Asignar propiedades. EL SET
nuevoObjeto["algunaLlave"] = "HD Magazine";
//Obtener propiedades. EL GET
var valor = nuevoObjeto["algunaLlave"]
```

Las otras dos maneras de crear son más elaboradas y son compatibles con el estándar ECMAScript 5[5].

```
//Usando Object.defineProperties
//Asignar propiedades. El SET
Object.defineProperties( nuevoObjeto, {
    "algunaLlave": {
          value: "HD Magazine",
          writable: true
    }
});
//Usando Object.defineProperty
//Asignar propiedades. EL SET
Object.defineProperty( nuevoObjeto, "nuevaLlave", {
    value: "HD Magazine",
    writable: true,
    enumerable: true,
    configurable: true
});
//Obtener propiedades. El GET
//Se pueden usar cualquiera de las dos primeras maneras
```

4 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object

5 http://kangax.github.io/es5-compat-table/

Constructores básicos

A pesar de que Javascript no soporta el concepto de clases, soporta funciones constructor especiales que trabajan con objetos. Esto se puede lograr solamente usando la palabra reservada “new”, y así llegar a tener un constructor.

Dentro de un constructor la palabra “this” referencia al nuevo objeto que esta siendo creado.

Un constructor básico luce de la siguiente manera:

```
function Revista( titulo, edicion, paginas ) {
  this.titulo = titulo;
  this.editorial = edicion;
  this.paginas = paginas;
  this.descripcion = function() {
       return 'La edicion ' + this.edicion + ' de ' + this.titulo + ' tiene ' + this.paginas+
' pags';
    }
}
```

Para poder usar este constructor, podemos hacerlo de la siguiente manera:

```
var revista1 = new Revista( 'HD Magazine', 'Cobra',  63)
var revista4 = new Revista('HD Magazine', 'Narciso', 74)
//Para mostrar las descripciones
console.log( revista1.descripcion() );
console.log( revista4.descripcion() );
```

Si bien es cierto el código anterior funciona como constructor, tiene algunos inconvenientes y no es tan óptimo como se quisiera, por suerte siempre hay mas de una manera de hacer las cosas.

Constructores con Prototypes

Toda función en Javascript tiene una propiedad llamada prototype, esta propiedad esta disponible cada vez que se crea un objeto también.

Sabiendo que podemos usar prototype[6], podemos tener el siguiente código funcionando y preparado para usar herencias, entre otras cosas, y obtener el mismo resultado anterior.

```
function Revista( titulo, edicion, paginas ) {
  this.titulo = titulo;
  this.edicion = edicion;
  this.paginas = paginas;
}
```

6 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/prototype

```
Revista.prototype.descripcion = function(){
    return 'La edicion' + this.edicion + ' de ' + this.titulo + ' tiene ' + this.paginas;
}

var revista1 = new Revista( 'HD Magazine', 'Cobra',  63);
var revista4 = new Revista('HD Magazine', 'Narciso', 74);

//Para mostrar las descripciones
console.log( revista1.descripcion() );
console.log( revista4.descripcion() );
```

De esta manera entonces se tiene una instancia del método descripción para todos los objetos Revista.

Happy Coding :)

VIM y el movimiento eficiente

El movimiento entre lineas de código en VIM, es básico para sacar partido a este editor tan particular. Veamos algunos trucos para hacer más eficientes nuestros movimientos.

Escrito por: **Pablo Bernardo** (PHP Developer)

Desarrollador Web especializado en **GNU/Linux, PHP, HTML, CSS y JavaScript**. Director y presentador del programa **#linuxIO** en **DesarrolloWeb.com**. Miembro Colaborador de la **Fundación Mozilla** y traductor en la **Mozilla Developer Network**. **Instructor de PHP** en escuelaIT. Usuario y promotor de GNU/Linux. VIM lover.

Webs:
Web personal: www.elkarmadelteclado.com

Redes sociales:
Twitter: **@voylinux**

Una de las particularidades que más llama la atención de cualquiera cuando se inicia en el uso de VIM como editor, es la manera en que navegamos a través de las lineas de texto. Vamos a ver algunos trucos para poder sacar más partido a la potencia de este *software* y dar un paso más en el camino de la eficiencia en la edición de código.

De las primeras cosas que aprendemos los usuarios de VIM, es a movernos utilizando **h j k l** como teclas principales en lugar de los cursores. Una de las ventajas de esto, es que podemos mantener los dedos siempre en la fila principal del teclado ahorrando desplazamientos y posibilitando la ejecución rápida de comandos del editor.

Existen muchos movimientos que podemos realizar utilizando los cursores, pero acostumbrarnos a los controles que propone VIM es fundamental para poder exprimir las posibilidades de otros trucos que se apoyan en estos.

Movimientos sencillos según VIM

Supongamos que estamos editando un archivo y queremos desplazarnos cuatro lineas más abajo. Tenemos varias opciones. Si sabemos que **j** es la tecla que usamos para desplazarnos hacia abajo, podemos pulsar **jjjj**. Pero evidentemente, hacerlo así no difiere mucho de pulsar cuatro veces el cursor hacia abajo.

Pero la filosofía del usuario de VIM es siempre ahorrar pulsaciones, ir directo al objetivo. Para esto, en el caso que nos ocupa, sabemos que queremos ir cuatro veces hacia abajo. En VIM, podemos indicar cuantas veces realizar un movimiento antes de llevarlo a cabo. Por lo tanto podríamos hacer directamente **4j** (cuatro veces hacia abajo). La misma filosofía es aplicable hacia arriba, evidentemente, sustituyendo **j** por **k**.

Otra opción es pulsar directamente el número de linea al que queremos ir, seguido de **G** o **gg**. Por ejemplo **12gg** para ir directo a la linea 12. O simplemente **:12** seguido de enter. Sencillo ¿verdad?. Veamos otros trucos de movimiento más avanzados, ya que estos forman parte de los más elementales de navegación.

> Para que VIM muestre los números de linea debes introducir el comando **`:set number`** o colocar **`set number`** en tu `.vimrc`

Algunas personas encuentran más cómodo editar código viendo el número de lineas que hay hacia arriba y hacia abajo, en lugar de la linea total respecto a su posición en el documento para moverse. Llamamos a esto números relativos. De esta manera, junto a cada linea verás un número que representa el número de líneas que la separan de aquella sobre la que te encuentras posicionado.

> Para que VIM resalte la linea sobre la que se encuentra el cursor, utiliza el comando **`:set cursorline`** o incluye la linea
> **`set cursorline`** en tu archivo `.vimrc` o **`set nocursorline`** para desactivarlo.

Copiando y pegando código

Otra cosa interesante en los movimientos de VIM es poder llevar a cabo acciones en un punto determinado del código sin tener siquiera que desplazarnos primero hasta allí. Esto forma parte de esa filosofía de VIM. ¿Qué forma hay más eficiente de hacer un movimiento que obtener el mismo resultado sin tener que moverse? Veamos como hacer esto desde el *normal mode* de VIM.

Imaginemos el siguiente ejemplo de código sencillos con sus números de línea.

```
1 function foo($foo){
2   if($foo == 'ejemplo'){
3      return TRUE;
4   }else{
5      return FALSE;
6   }
7  }
8
9  function otrofoo($foo){
10
```

Supongamos queremos escribir la misma condición en la linea 10. ¿Escribiríamos de nuevo todo el código?. No. Si no es necesario. Recordemos la filosofía de ahorrar pulsaciones de teclas.

Si queremos copiar la linea 2 y pegarla en la 10, de nuevo tenemos varias opciones. Podemos desplazarnos con **k** hasta la linea 2, pulsar **yy** para copiar la linea entera, navegar con **j** hasta la linea 9 y pulsar **p** para pegar. Pero...poco eficiente ¿no?. Podemos hacer esto en cualquier editor. Utilizando los comandos adecuados de VIM, podemos realizar esto sin mover el cursor ni estar siquiera en la linea 9.

La manera de hacer esto es utilizar el comando **`:copy`** (o su abreviatura **`co`** o su alias **`:t`**).
El comando **`:copy`** admite una linea (o rango de ellas) como origen y otra linea como destino (o **`.`** para pegar bajo la linea actual). Así en nuestro ejemplo, escribiríamos **`:2copy9`** para copiar la linea 2 y pegarla bajo 9 (podemos sustituir **`copy`** por **`co`** o **`t`**). ¿Y si queremos copiar todo el condicional?. Sencillo.

Hemos comentado que **`:copy`** admite un rango de lineas como origen. Los números de linea del rango se introducen separados por una coma, en nuestro ejemplo sería **`:2,6co9`** para copiar todo el texto entre las lineas 2 y 6, ambas inclusive y pegarlo bajo la linea 9.

Todos estos atajos y comandos pueden parecer confusos al principio, pero según se van incorporando a la rutina de edición, se incorporan como algo tan natural para desarrollador como cualquier simple atajo de teclado.

Moviéndose por la ventana (algunos comandos rápidos)

En ocasiones, los movimientos que queremos realizar no son relativos a líneas de código sino a la posición en la ventana. Para eso tenemos algunos comandos sencillos.

Para desplazar el cursor hasta la mitad de la ventana sin depender del número de línea, puedes utilizar el comando **M** (en mayúscula) para moverte hasta el medio (**M**iddle). ¿Quieres ir hasta la parte superior de la ventana ? Pulsa **H** para ir hasta la parte superior de la ventana (**H**ome) o **L** para ir a la última linea de la ventana (**L**ast line).

> Si quieres ver los números de linea relativos a la navegación desde la linea actual, introduce en VIM **`:set relativenumber`**. Los números de lineas superiores e inferiores indicarán la distancia en lineas desde la actual.

Incorporando nuevos comandos

En posteriores artículos aprenderemos otros trucos y configuraciones que podemos ir incorporando a nuestra rutina de edición con VIM y aprenderemos a manejarnos con los objetos de texto, configuraciones por tipos de archivo y otras opciones.

Si no estás muy acostumbrado aún al trabajo con VIM, mi consejo es que no te dejes asustar por su potencia. La razón por la que mucha gente lo encuentra difícil, es porque intenta aprender demasiadas cosas a la vez.

Incorpora poco a poco los trucos que vas aprendiendo, no intentes dominar mas de un par de ellos por semana. Si lo aprendes de esta manera progresiva, te asombrarás de lo rápido que pasan a formar parte de tu manera enfrentarte a la edición de código y te harán entender por qué tantos desarrolladores consideran el aprendizaje del manejo de VIM como una gran inversión.

Para aprender movimientos más básicos o los fundamentos del manejo de VIM, ejecuta en una terminal el comando **`vimtutor`** . VIM te mostrará su propio tutorial desde cero, bastante completo y traducido al español

Sudoers: Cambiando de identidad

Las empresas manejan sistemas *NIX para gestionar ciertos recursos y, generalmente, esta gestión es llevada a cabo por un número concreto de personas, las cuales no deben poder acceder a los mismos recursos. Por esta razón presentamos a los *sudoers*.

Escrito por: **Pablo González Pérez** (Project Manager)

Ingeniero Técnico en Informática de Sistemas e Ingeniero en Informática por URJC. Project Manager en 11Paths, empresa perteneciente a Telefónica Digital. Premio Extraordinario Fin de Carrera por la URJC en Ingeniería Técnica en Informática de Sistemas en 2009. Premio al mejor expediente de su promoción por la Escuela Técnica Superior de Ingeniería Informática de la URJC en 2009. Fundador de Flu-Project. Autor de: Metasploit para Pentesters y Pentesting con Kali.

Webs:
www.flu-project.com

Redes sociales:
Twitter: **@pablogonzalezpe**

En un entorno típico de empresa con diversos servidores y máquinas, donde dichos entornos son gestionados por un alto número de operadores, puede llegar a ser confuso, complejo y sobretodo inseguro el proporcionar un acceso total a una máquina. En el instante que un administrador proporciona una credencial de *root* a un grupo de operadores encargados de llevar a cabo la administración de ciertas máquinas, se está abriendo una brecha de seguridad importante en la empresa ya que dicha autorización pasará por ciertas máquinas y por ciertos usuarios, perdiendo el control sobre quién posee realmente dicha credencial.

Hoy en día es muy factible disponer en los servidores de cualquier entorno, producción, desarrollo, pre-producción, etcétera, un sistema de delegación de identidad y privilegios, el cual proporciona un acceso parcial o completo a las máquinas tomando las identidades de otros usuarios, siempre y cuando el administrador así lo haya configurado. Este sistema es denominado sudo o *Super User Do*.

Este sistema tiene como principal funcionalidad proporcionar a un usuario la posibilidad de realizar tareas en nombre de otro usuario, es decir, con los privilegios de éste último. Es importante diferenciar que un usuario que puede utilizar sudo, no tiene porqué ser para utilizar o tomar la identidad del *root*. En otras palabras, el usuario A, siempre y cuando así esté configurado, podría tomar la identidad del usuario B, y que éste no sea el usuario *root*. El objetivo de sudo es que el usuario realice solo las acciones que necesite con la identidad de otro usuario, que será el que realmente tenga los privilegios para hacerlo. Con esta solución estamos minimizando el impacto de delegar una credencial con acceso total a la máquina.

Generalmente, un usuario que comienza a utilizar sudo no configura el fichero dónde se almacenan las directivas y políticas, con todo el potencial que sudo pone al alcance del administrador. En estas directivas se especifican que acciones puede llevar a cabo cada usuario que pertenezca a los *sudoers*. Una buena práctica será utilizar la técnica como restrictivo por omisión, es decir, denegar el uso del cambio de identidad salvo a aquellos usuarios que lo necesiten.

Además, cada usuario que necesite utilizar sudo, deberá disponer específicamente en el fichero de configuración las acciones permitidas, y no más que las que explícitamente se indiquen en dicho fichero.

Configuración de sudoers

El fichero para la configuración de esta característica de seguridad se encuentra en la ruta */etc/sudoers*. Una propiedad fundamental del fichero es que agrupa secciones que permiten al administrador una mejor configuración. Para acceder al archivo se debe utilizar el comando *visudo*, con el que bloquea el archivo evitando que otro usuario en paralelo pudiera editarlo. Además, cuando se utiliza este comando, una vez finalizada la edición se comprueban errores sintácticos en la manipulación del fichero *sudoers* y de esta manera evitar que la aplicación quede inestable.

Hay elementos de interés dentro del archivo de configuración de *sudoers* como es la cláusula opciones (*Defaults*). Estas opciones sirven para modificar el comportamiento de *sudo*. El valor de dichas opciones son booleano, valores numéricos o *strings*. Se pueden asignar a nivel global, es decir, a todos los usuarios que se encuentren especificados como *sudoers* o a nivel particular, un usuario en concreto. A continuación se muestra un pequeño ejemplo para simplificar su comprensión.

```
Defaults        env_reset, timestamp_timeout=5, passwd_timeout=1, passwd_tries=2
Defaults:pablo insults
```

Se puede diferenciar las opciones globales y las opciones particulares. En la parte global se especifica un *timeout* para introducir la password de un minuto, cinco minutos de validez de sesión hasta que se vuelva a solicitar la contraseña en una nueva acción de *sudo* y dos intentos para introducir correctamente la contraseña. En el caso del usuario particular se indica que si falla en la autenticación se mostrará por pantalla una serie de frases "curiosas".

Por otro lado, los alias nos permites agrupar acciones, usuarios, entornos bajo el mismo nombre. Hay distintos tipos de alias como por ejemplo *Host_Alias, User_Alias* y *Cmnd_Alias*, los cuales son bastante intuitivos.

Por último, el elemento más importante ya que es el que realiza la concesión de identidad o no, es la ACL. Son las reglas que el administrador definirá para proporcionar la toma de una identidad a un usuario o no. La sintaxis es la siguiente *<usuario/%grupo> <host> = [(<runAsUserList> :<runAsGroupList>)] <comando1>, ..., <comando N>.*

```
root    ALL=(ALL:ALL) ALL
pablo ALL = /bin/date
```

Lo que se indica en estas líneas es que el *root* puede ejecutar bajo cualquier identidad cualquier acción, mientras que el usuario pablo puede ejecutar tomando la identidad de *root,* ya que no se especifica entre paréntesis ni usuarios ni grupos, el comando *date* y ninguna acción más.

El comando *sudo* dispone de unos parámetros que son interesantes para aprovechar su potencial. A continuación se presenta un breve resumen:

- *-u*. Indicará a *sudo* que identidad se quiere tomar, si se tiene configurado en el archivo que se puede tener. Si no se especifica el parámetro, se intentará ejecutar la acción con la identidad de *root* por defecto.
- *-k.* Este parámetro invalida una sesión anterior de *sudo*.
- *-l*. Informa al usuario de sus privilegios como *sudoer*.

Ejemplo final del archivo

En el siguiente ejemplo final de archivo de *sudoers* se configura dos usuarios bajo el alias TECNICOS y otro usuario será el administrador y pertenecerá al grupo de administradores. Además, se configura bajo un alias los comandos que los técnicos podrán utilizar. Se puede entender que los técnicos sólo podrán realizar ciertas acciones bajo la identidad del administrador, mientras que éste podrá ejecutar cualquier acción.

```
Defaults        env_reset,timestamp_timeout=5,passwd_timeout=1,passwd_tries=1
Defaults:pablo insults
# Host alias specification
# Ninguno
# User alias specification
User_Alias TECNICOS = user1,pablo
Cmnd_alias BARRABIN = /bin/
# User privilege specification
root    ALL=(ALL:ALL) ALL
TECNICOS ALL = BARRABIN
# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL
# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) ALL
```

Como conclusión añadir que la delegación de privilegios en ciertas acciones a través de *sudo* permite obtener una capa de seguridad y de control que no dispone a través de la delegación de credenciales genérica.

Sudo no deja de ser un *software* el cual tiene sus versiones y puede tener implementados fallos explotables, por ejemplo últimamente se ha conocido alguno grave que permitía la escalada de privilegios en el sistema, pero manteniendo *sudo* actualizado la delegación de identidades por acciones es altamente segura.

La terminal en GNU/Linux: "Back to the future?"

Si piensas la terminal es algo del pasado o no necesitas aprender a usarla, es hora de viajar al futuro a través de la linea de comandos.

Escrito por: **Pablo Bernardo** (Miembro Colaborador de la Fundación Mozilla)

Desarrollador Web especializado en **GNU/Linux, PHP, HTML, CSS y JavaScript**. Director y presentador del programa **#linuxIO** en **DesarrolloWeb.com**. Miembro Colaborador de la **Fundación Mozilla** y traductor en la **Mozilla Developer Network**. **Instructor de PHP** en escuelaIT. Usuario y promotor de GNU/Linux. VIM lover.

Webs:
Web personal: http://www.elkarmadelteclado.com

Redes sociales:
Twitter: **@voylinux**

Las personas que no son usuarios habituales de sistemas GNU/Linux suelen como mínimo sorprenderse del uso tan habitual que hacemos de la terminal. Algunos comentan que es un sistema arcaico, o solo para administradores de sistemas o incluso "hacer las cosas de la manera más difícil". Lo que parece curioso es que hoy en día esto ocurra ente desarrolladores.

Las interfaces gráficas no son un problema para el usuario de GNU/Linux, no se trata de que estemos en contra. Para muchas tareas son una manera rápida de llevarlas a cabo si nos encontramos en ese momento realizando otra tarea en el escritorio. El problema surge cuando dejamos de ser conscientes de que la interfaz es solo una abstracción de lo que realmente ocurre en el sistema y perdemos la perspectiva de que sus posibilidades están limitadas para muchas tareas.

Para empezar y por si el lector no lo supiese, conviene aclarar que lo que abrimos cuando accedemos a la terminal desde nuestro entorno de escritorio no es una verdadera terminal, sino un emulador de la misma, que nos permite realizar las mismas tareas, pero de nuevo depende del entorno gráfico para funcionar. Es como abrir una "ventana" en el escritorio que nos permite acceder a lo que ocurre debajo, en el verdadero sistema que se encuentra detrás de los iconos y colores.

La terminal de tu sistema, funciona de manera independiente al entorno gráfico. Es más, es la manera en que podrás seguir accediendo al sistema en caso de que este entorno gráfico presente algún problema o incluso si no existe.

Hablemos un poco del poder que uno adquiere cuando hace uso de la terminal.

Si desde el escritorio queremos realizar una tarea, la que sea, sin acceder a la consola, dependemos obviamente de las opciones que el desarrollador haya incluido en el programa que queremos utilizar para llevar nuestro propósito a buen puerto. Sin embargo, para cualquier tarea que el sistema pueda realizar, podríamos acceder a la terminal sin depender de que el software lo haya implementado o no en su interfaz entre las opciones que nos ofrece como usuarios. Si el sistema puede hacerlo, desde la terminal podemos hacerlo, así de simple y así de poderoso.

Tareas como copia o movimiento de archivos, configuraciones simples o complejas del sistema, instalaciones de software o programación de tareas, son solo algunas de las tareas que desde la terminal es posible implementar en una o dos lineas de comandos, mientras que en el entorno gráfico pueden ser algo más complicadas o implicar la utilización de diferentes programas.

El uso de la terminal permite al usuario no solo navegar por el sistema de archivos de una forma mucho más directa y eficaz, sino llevar a cabo todo tipo de labores de mantenimiento e interacción incluso fuera de nuestra propia máquina. Ya sea la conexión a servidores externos, descarga de archivos, envío de datos o control remoto de otros equipos.

Hay varias cosas que se pueden comentar en el caso de los desarrolladores que opinan que se trata de una herramienta antigua, que no necesitan aprender a utilizarla o que yo en día todo puede hacerse de otra manera. Especialmente a los que son usuarios de GNU/Linux pero mantienen estas opiniones.

En primer lugar, tened en cuenta que los entornos de escritorio varían en función de la distribución o de lo que nosotros mismo hayamos decidido instalar, mientras que la terminal está presente en todos los sistemas GNU/Linux. ¿Qué te proporciona esto?. Muy sencillo: libertad.

Conocer a fondo lo que puedes hacer desde la terminal te hace cada vez más libre en el uso de la computadora porque te vuelve independiente. Te permite hacer lo que quieres y sabes hacer sin necesidad de encontrarte en un sistema ni un escritorio determinados.

Otra cosa a desatacar para los desarrolladores web, por ejemplo, es que no todos los servidores con los que se encontrarán en su carrera serán servidores compartidos. Esto parece obvio pero añade una variable importante en la decisión de comenzar a conocer el poder y las posibilidades de la terminal. La mayoría de los servidores con los que tendrán que trabajar, es probable que ni siquiera cuenten con un entorno gráfico instalado y por tanto, la única manera de trabajar será a través de la linea de comandos.

Es posible que el programador piense también que esto es trabajo de la gente de sistemas y en parte es cierto, pero muchas de las tareas mínimas que tendrá que llevar a cabo dentro del servidor, le resultarán mucho más cómodas y llevaderas si sabe moverse con soltura a través de la linea de comandos. Incluso para sus tareas de desarrollo en su máquina local, será mucho más eficaz si sabe simular con la mayor exactitud posible, el trabajo en el servidor.

Otro dato que debería mover al desarrollador al uso más a menudo de la linea de comandos, es que es muy buena forma de mejorar su conocimiento del sistema de permisos y el sistema de archivos de la máquina, lo que ayudará a implementar mejor cualquier tarea que tenga que ver con estas cuestiones dentro de sus programas.

Pero aún hay más cosas que apuntan directamente al panorama actual y futuro del desarrollo de software.

Sistemas de control de versiones, frameworks en JavaScript para mejorar nuestro flujo de trabajo, preprocesadores CSS, sistemas para minificar nuestros archivos, frameworks para el desarrollo de aplicaciones, servidores en diferentes lenguajes, ... tecnologías que se están volviendo cada día más comunes para los desarrolladores web. Para todas ellas no siempre existe un software que nos permita trabajar de una manera cómoda, especialmente en GNU/Linux. Quizá el software que exista no permita sacar todo el partido posible a la tecnología , no está disponible para nuestra distribución o simplemente no existe una herramienta gráfica para utilizarlo. Si dependemos del entrono gráfico estamos atados.

Pero lo que mucha gente desconoce o simple desaprovecha, es que normalmente estos sistemas incorporan todas sus opciones a través de una interfaz de linea de comandos o CLI (Command Line Interface). Por tanto, una vez más, la manera de sacarle todo el jugo a la tecnología sin depender de lo que se haya implementado a través de botones o pantallas es la linea de comandos. Solo esta nos garantiza poder implementar todas las opciones que el software tiene disponible.

Por estas y muchas más razones podemos afirmar que el uso de la terminal y la linea de comandos no solo no son opciones del pasado, sino que están plenamente vigentes. Es más, cada vez más tecnologías incorporan la posibilidad de ayudarnos en nuestras tareas como programadores haciendo eso de herramientas CLI.

Si eres usuario de GNU/Linux abre ahora la terminal y comienza con una tarea sencilla y ve incorporando el uso de la misma poco a poco. Deja de menospreciar el uso de la terminal y pensar que es algo del pasado. Vuelve más bien al presente, aprende a ser un usuario libre y prepárate para el futuro.

HACKERS & DEVELOPERS MAGAZINE

PRESENTA

mozilla zone

con el apoyo de:

mozilla FOUNDATION

Persona: El login libre de Mozilla

Vamos conocer Persona, el sistema de Login seguro y libre que se lo pone fácil a desarrolladores y usuarios.

Escrito por: **Pablo Bernardo** (Miembro Colaborador de la Fundación Mozilla)

Desarrollador Web especializado en **GNU/Linux, PHP, HTML, CSS y JavaScript**. Director y presentador del programa **#linuxIO** en **DesarrolloWeb.com**. Miembro Colaborador de la **Fundación Mozilla** y traductor en la **Mozilla Developer Network**. **Instructor de PHP** en escuelaIT. Usuario y promotor de GNU/Linux. VIM lover.

Webs:
Web personal: http://www.elkarmadelteclado.com

Redes sociales:
Twitter: **@voylinux**

El sistema de registro y verificación de usuarios es una problema muy habitual en el desarrollo de software. Pero no es un problema únicamente para los desarrolladores, sino que representa además una molestia para los usuarios.

A nivel de código, es complicado tener que controlar todas las posibles variables derivadas de la identificación, el registro y algo muy importante: la seguridad y fortaleza de la contraseñas generadas a través de la aplicación. Del lado del usuario, cada vez se generan más cuentas, contraseñas y diferentes identidades a través de todas los servicios web en los que nos registramos o necesitamos identificarnos.

Es aquí donde aparece una tecnología de las que demuestra que Mozilla se preocupa realmente por hacer un internet más libre y seguro. Mozilla es mucho más que Firefox y vamos a conocer una de sus soluciones a nivel de código: Persona.

Persona es un sistema libre y distribuido de autenticación para aplicaciones web, que se construye sobre la base del protocolo BrowserID. Elimina por completo la necesidad por parte de los desarrolladores de comprobar en su propia aplicación la identidad de los usuarios. ¿Cómo es posible?. Vamos a explicarlo de manera sencilla.

Cuando el usuario se encuentra con una página web que implementa el sistema Persona y quiere identificarse, simplemente ha de introducir su correo electrónico. ¿Por qué el correo?. Primero porque es una dato que todo el mundo recuerda, no como cientos de nombres de usuario únicos que se generan en las diferentes aplicaciones.

Pero además porque esto hace posible la verificación de la identidad no en la propia web, sino ante el proveedor de correo.

Básicamente la página solicita al usuario su mail y a partir de ahí el usuario, a través de una interfaz común y reconocible para todos los sitios que lo implementen, se comunica directamente con su proveedor de correo, que es quien le solicita la contraseña y la comprueba.

Esto último es un dato muy importante que añade un componente de seguridad. El sitio web en ningún momento recoge la clave, ni puede rastrear estos datos del usuario. La web simplemente recibe a través de Persona una respuesta positiva o negativa de autenticidad de los datos. Así de sencillo, la aplicación solo tiene que "escuchar" la respuesta y reaccionar según la misma.

Pero ¿Cómo es el proceso de implementación de Persona a nivel de código?. Muy sencillo.

Para empezar, por supuesto , Mozilla proporciona directamente el archivo JS con la librería que implementa Persona. Es tan fácil como incluirla en la pagina.

```
Por cuestión de versiones y desarrollo, se recomienda incluirlo siempre de la url remota, no descargar el archivo JS en local.
<script src="https://login.persona.org/include.js"></script>
```

Solo con esto ya tenemos a nuestra disposición todos los métodos del API de Persona.
A partir de aquí, podemos crear un botón de Login y uno de Logout y en JavaScript capturar los eventos click de ambos. Según se trate del botón de Login o Logout, en su evento click haremos la llamada a los métodos navigator.id.request() o navigator.id.logout() respectivamente.

Mozilla proporciona también el recurso del aspecto que puedes incorporar para que los botones sean reconocibles por el usuario en este enlace:

https://developer.mozilla.org/es/docs/persona/branding[7]

A continuación, solo tenemos que llamar al método navigator.id.watch() en JavaScript y dentro del mismo, referenciar al usuario actual y decidir las acciones a realizar cuando todo sea correcto, cuando no o cuando se produzca el Logout.

Para ello debemos tener en cuenta que debemos enviar una petición al servidor porque aún se deben verificar los datos para comprobar la respuesta del sistema al intento de login del usuario. Hasta ahora hemos llamado al API, pero necesitamos que el sistema verifique la repsuesta. Realizamos esto del lado del servidor para estar seguros de que no se modifica la petición a través de JavaScript.

7 https://developer.mozilla.org/es/docs/persona/branding

Desde la página de nuestro servidor a la que enviemos la petición, podemos solicitar confirmación al sistema de verificación de Mozilla, que realizará el tramite y nos devolverá un sencillo JSON con la respuesta. Este JSON, evidentemente no contiene datos acerca de las claves del usuario, sino algo del tipo:

```
{
  "status": "okay",
  "email": "voylinux@ejemplo.com",
  "audience": "https://ejemplo.com:443",
  "expires": 1308859352261,
  "issuer": "ejemplo.com"
}
```

Esto JSON, como ves contiene datos de la correcta verificación, pero no datos personales de ningún tipo. Solo el mail, que es nuestro identificador o nombre de usuario.

También cabe destacar, que el método navigator.id.watch() del que antes hablamos debería estar presente en todas la paginas de la web, de otra manera no es posible gestionar de manera correcta el control de usuarios. Lo más sencillo es incluir todo esto en el archivo general de JavaScript de nuestra página.

Algo que debemos entender es que este sistema nos ayuda en la verificación de usuarios, pero no impide que los mismos puedan tener una ficha con sus datos, preferencias y demás una vez logueados. Es solo el sistema de nombre de usuario y contraseña, así como su comprobación lo que estamos sustituyendo.

Puedes encontrar ejemplos detallados de código en la documentación oficial de Mozilla[8], así como documentación técnica detallada pero como ves, se trata de un sistema sencillo que en unas pocas lineas de código, proporciona un sistema seguro, cómodo y probado de solucionar un problema común a muchas aplicaciones.

Mi intención es, si no la conocías, descubrirte la posibilidades de esta posibilidad que Mozilla nos ofrece a los desarrolladores.

Como desarrollador, tienes la oportunidad de implementar una tecnología que no solo te lo pone fácil para controlar tu gestión de login, sino que incorpora un sistema fácil de usar y confiable para tus usuario. Y también la oportunidad de contribuir con Mozilla a crear la web que todos necesitamos, una web que pueda ser cómoda para quienes la utilizan, sin tener que renunciar a su seguridad y privacidad.

8 https://developer.mozilla.org/es/docs/Persona/Quick_Setup

INFORMÁTICA & Derecho

Licencias del Software Libre y Libertades del Usuario

Para abocarse al estudio particular de las licencias del software libre, antes es necesario abordar y entender las fuentes éticas en las que se fundamenta su existencia y aplicación jurídica, pues todas coinciden en punto indiscutible que debe asegurarse: la libertad del usuario.

Escrito por: **Silvio Messina** (Desarrollador Web - Est. de Derecho, Universidad de Buenos Aires)

Silvio es **Desarrollador**, **Profesor en Tecnologias Web**, Wordpressero y Pre-Abogado (o estudiante de Derecho) orientado al estudio del Copyright, Propiedad Intelectual, Software y Cultura Hacker. Defensor de la Educación abierta, el auto desarrollo intelectual y el libre intercambio.

Webs:
Web personal: www.silviomessina.com

Redes sociales:
Twitter: **@smessina_**

Quizás uno de sus mejores aportes de las licencias libres es el considerar al usuario un sujeto de derecho en la relación entre la obra y su autor, y como este usuario a su vez puede generar una relación propia con la obra; alimentando el espíritu que ha impulsado toda esta corriente de pensamiento: la libertad.

Licencias en el desarrollo de Software

La aplicación de licencias al software tiene por objetivo reglamentar los derechos que se pueden ejercer, siendo los principales los derechos de uso y de distribución. Las licencias libres hacen foco en establecer las libertades que el desarrollador puede otorgar al usuario y las que éste puede ejercer frente a dichos dos derechos:

```
Libertad #0: facultad de usar y ejecutar el software para cualquier propósito.
Libertad #1: facultad de estudiar y modificar el software a las necesidades de uno.
Libertad #2: facultad de copiar y distribuir el software.
Libertad #3: facultad de distribuir las versiones modificadas del software.
```

Un Software es considerado Libre si los usuarios tienen todas estas libertades, es decir; que se tiene absoluta capacidad y autonomía de acción.

Las mismas representan un paradigma distinto del derecho de autor, pues no desconocen el elemento "autoría" del desarrollador sobre el software, ya que ofrecen mecanismos de disposición hacia los usuarios que revaliden su carácter de autor; y estos a su vez puedan beneficiarse y beneficiar a los demás manteniendo el régimen no restrictivo que recibieron originalmente del autor y su obra "libre".

Entendiendo las cuatro libertades de los usuarios

#0 - Libertad de uso
Esta es la libertad más básica perseguida, ya que ofrece al usuario que obtiene una copia del programa el derecho de usarlo con cualquier propósito. Ejercer esta libertad no se limita solo a "usar", sino también a la finalidad o propósito perseguido por tal uso. Además implica que no se requiere la autorización del autor para ejercer tal uso o propósito.

#1 - Libertad de estudio y modificación
Por excelencia es la libertad más ejercida en el ámbito del desarrollo de software. Esta libertad se sustenta en la obligación del programador (autor original) de poner a disposición el código fuente completo del programa, ofrecido de un modo legible para su estudio y para que pueda ser modificado efectivamente.

A primera vista, esta libertad beneficia más a un desarrollador o programador que a un usuario final, pero incluso este usuario final puede beneficiarse al hacerse con el programa y encomendar por su cuenta a un programador que lo estudie y modifique para adaptarlo a las necesidades del primero.

Esta libertad tiene un límite: el estudio y modificación del software solo puede hacerse para "uno mismo", es decir que se podrá practicar en tanto y en cuanto la utilidad que se haga del software modificado (fundándose en la **libertad #0**) sólo se realice para satisfacer la necesidad en la que se basa ejercer esta **libertad #1**.

Se puede afirmar entonces que, tanto la **libertad #0** como la **libertad #1** están pensadas para el ámbito privado del usuario y no se ocupan de abordar sobre la distribución pública del propio software como de las modificaciones que se le hagan, las cuales encuentran respuesta en las **libertades #2** y **#3**.

#2 - Libertad de distribución
Caracterizada por ser la libertad que abre las puertas al circuito de distribución masivo, otorgando la facultad al usuario de redistribuir copias del software obtenido, tanto en su versión compilada como su código fuente. La misma libertad determina que no se puede restringir la difusión del software a algunos países, ni tampoco añadir restricciones que nieguen a los demás las mismas libertades que quien distribuye ya tiene. El usuario que participe de este circuito de distribución puede hacerlo de forma gratuita o cobrando por la misma, a cualquiera y en cualquier lugar.

Es ampliamente discutida la cuestión de cobrar por la distribución de software, pues se tiene la creencia que se está lucrando con el trabajo de otro. La **Free Software Foundation** da el visto bueno a cobrar por la distribución de software libre, pues constituye un modo de dar a conocer los proyectos, y por supuesto el usuario que busca acceder a dicho software siempre tendrá la opción de pagar por el mismo u obtenerlo por otras vías.

En concreto, la **libertad #2** está confeccionada para otorgar el derecho a los usuarios de formar parte del circuito de distribución, e imponer a los autores la obligación de abstenerse a restringir las vías y modos de distribución que cualquier usuario pueda ejercer.

#3 - Libertad de distribución de versiones modificadas
Identificada como la libertad que más nutre al espíritu colaboracionista de los desarrolladores. En cierta forma esta libertad surge como una combinación de las otras tres, pues gracias a ejercer la **libertad #0** y **#1** se obtiene como resultado una versión modificada del programa obtenido.

Es lógico que si la **libertad #2** autoriza al usuario a redistribuir el software del mismo modo que tuvo acceso, no puede quedar por fuera hacerlo con las mejoras aplicadas al entorno privado. A partir de este punto, se es libre de hacer la distribución modificada, de modo gratuito o comercial; siguiendo los parámetros de la **libertad #2** en donde lo esencial es asegurarle al usuario ejercer las cuatro libertades.

Copyleft, moviendo el derecho de autor a la izquierda

El copyleft[1] es un modo de ejercer el copyright “a la inversa”[2], un mecanismo para que sobre un programa libre, se exija que todas las versiones modificadas y extendidas también sean libres. Resulta ser el medio por el cual legalmente se hace valer la voluntad del programador de asegurar las cuatro libertades y que nadie que se encuentre participando del proceso de redistribución altere a los nuevos usuarios las libertades obtenidas.

Cabe destacar que copyright y copyleft son dos caras de una misma moneda, ya que ambas buscan proteger la propiedad intelectual e imponen restricciones sobre el uso y distribución de contenidos. Bajo un sentido menos restrictivo que el copyright, el copyleft garantiza que se preserven estas libertades para cualquier receptor de una copia, o de una versión derivada.

En resumen...

Las cuatro libertades son el mayor aporte jurídico del software libre y la manifestación de toda una visión ética de cómo debería ser una realidad social más equilibrada; donde no existan restricciones en detrimento de las libertades reconocidas. El "que” del software libre está más que claro, son sus cuatro libertades, el "como" hacer que esas libertades se aseguren, garanticen y respeten sera tarea de las licencias y su aplicación.

Nos vemos en la próxima entrega.

1 http://www.gnu.org/copyleft/copyleft.es.html / http://es.wikipedia.org/wiki/GNU_General_Public_License#Copyleft
2 http://es.wikipedia.org/wiki/Copyleft#M.C3.A9todos_de_aplicaci.C3.B3n