

AÑO ----- 0
NÚMERO ----- 4
FECHA: 2013-02-25

#4

“Narciso”

HD

Hackers & DEVELOPERS

Magazine digital de distribución
mensual sobre Software Libre, Hacking y Programación
para profesionales del sector de Tecnologías de la Información

Staff

Eugenia Bahit	Arquitecta GLAP & Agile Coach
Índira Burga	Ingeniera de Sistemas
Laura Mora	Administradora de Redes y Sistemas
María José Montes Díaz	Técnica en Informática de Gestión
Milagros Infante Montero	Est. Ingeniería de Sistemas



Hackers & Developers Magazine se distribuye bajo una licencia **Creative Commons Atribución NoComercial CompartirIgual 3.0 Unported**. Eres libre de copiar, distribuir y compartir este material.
FREE AS IN FREEDOM!

Hackers & Developers Magazine, es una iniciativa sin fines de lucro destinada al fomento y difusión de las tecnologías libres presentes o futuras, bajo una clara óptica docente y altruista, que resulte de interés técnico y/o científico a profesionales del sector de Tecnologías de la Información. Hackers & Developers Magazine se sostiene económicamente con el apoyo de la comunidad, no recibiendo subvención alguna de ninguna empresa, organización u organismo de Gobierno. Necesitamos de tu apoyo para poder mantener este proyecto.

Ayúdanos a continuar con este proyecto

Puedes hacer un donativo ahora, de 10, 15, 25, 50, 100 o 150 USD para ayudar a que Hackers & Developers Magazine pueda seguir publicándose de forma gratuita, todos los meses. Puedes donar con PayPal o Tarjeta de Crédito a través del siguiente enlace:

www.hdmagazine.org/donar

CON TU DONACIÓN DE USD 150
RECIBES DE REGALO,
UNA FUNDA DE
NEOPRENE PARA TU
ORDENADOR PORTÁTIL
VALUADA EN USD 25.-
(Origen: Estados Unidos)



“Hacker es alguien que disfruta jugando con la inteligencia”

Richard Stallman
Free Software, Free Society
(Pág. 97), GNU Press 2010-2012

En esta edición:

PEP8 de Python.....	4
Hal 9000 Junior. (Primera Parte).....	9
Pylint al rescate.....	13
Manual de MVC: (3) Controladores.....	17
Double Test con ZendFramework2	23
¿Cómo crear aplicaciones Web PHP con EuropioEngine?.....	27
Manual de Perl (Parte II).....	41
Conociendo a DOM: Parte II.....	49
Pásate a GNU/Linux con Arch: Pacman, el gestor de paquetes.....	54
Explorando Mosh.....	64
Agilismo en palabras simples.....	67
De estudiante a programador.....	72

Y LAS SECCIONES DE SIEMPRE:

ASCII Art.....	Pág. 73
Este mes: «The first Daffodil of Spring» by Susie Oviatt	
Zona U!.....	Pág. 74
La comunidad de nuestros lectores y lectoras	

Créditos

Hackers & Developers Magazine es posible gracias al compromiso de:

Responsable de Proyecto
Eugenia Bahit

Responsables de Comunicación

Indira Burga (Atención al Lector) - Milagros Infante (Difusión)

Staff Permanente

Eugenia Bahit
Arquitecta GLAMP & Agile Coach
www.eugeniabahit.com

Indira Burga
Ingeniera de Sistemas
about.me/indirabm

Milagros Infante Montero
Estudiante de Ingeniería en Sistemas
www.milale.net

Laura Mora
Administradora de Redes y
Sistemas GNU/Linux
blackhold.nusepas.com

María José Montes Díaz
Técnica en Informática de Gestión
archninf.blogspot.com.es

Colaboradores Estables

Elizabeth Ramírez
(Ingeniera Electrónica)

Sergio Infante Montero
(Ingeniero Informático)

Yecely Díaz
(Maestra en Inteligencia Artificial)

Redactores Voluntarios

Celia Cintas
Eliana Caraballo

Difusión

Hackers & Developers Magazine agradece a los portales que nos ayudan con la difusión del proyecto:



www.debianhackers.net



www.desarrolloweb.com



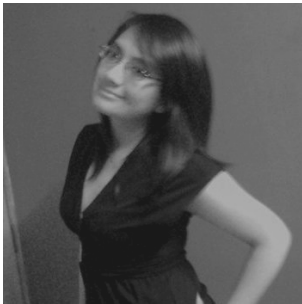
www.desdelinux.net

E-mail de Contacto:
contacto@hdmagazine.org

PEP8 de Python

Qué mejor que contar con un documento que liste las convenciones que tiene el lenguaje que usamos para desarrollar. PEP8 es la guía de estilo de Python. La clave para este PEP es una de las ideas de Guido Van Rossum, que el código se suele leer mucho más de lo que se escribe.

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



Estudiante de Ingeniería Informática. Miembro de **APESOL** ([Asociación Peruana de Software Libre](#)) y 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 / Identi.ca: [@milale](#)

La PEP (*Python Enhancement Proposal*) es la propuesta de mejora de Python, considerada como estándares. Son las guías de estilo que se encuentran en la PEP número 8, que están dirigidas a mejorar la legibilidad del código y a lograr consistencia. Esto último es fundamental dentro de un módulo o función, pero también es muy importante saber cuando ser inconsistente. Por ejemplo, podemos toparnos con una situación en la que debemos saber diferenciar si al aplicar alguna regla el código será menos legible.

“Una consistencia estúpida es el duende de las mentes pequeñas.” - Ralph Waldo Emerson

Es importante recordar que PEP nos dice que > “La legibilidad cuenta”, es importante seguir buenas prácticas de programación y al final que mejor que lograr un código limpio que cumpla todos los estándares.

Formato del código

Indentación:

Se debe usar 4 espacios por cada nivel de indentación. Cuando se indente código no se deben mezclar tabuladores o espacios; siempre es recomendable solo hacerlo con espacios.

Tamaño máximo de línea

Todas las líneas deben contar con un máximo de 79 caracteres y de 72 caracteres a líneas empleadas para documentación o comentarios. Al dividir líneas largas se suele continuar la línea de forma implícita dentro de paréntesis, corchetes o llaves, pero queda mejor el uso de una barra invertida (sobre todo, cuando no exista posibilidad de agrupamiento): \

```
def __init__(self, first, second, third,
             fourth, fifth, sixth):
    output = (first + second + third
             + fourth + fifth + sixth)
```

Líneas en blanco

Las líneas en blanco separan las funciones no anidadas y definiciones de clases (con dos líneas) y las definiciones de métodos dentro de una misma clase (con una línea), se pueden usar líneas en blanco extra; python acepta el carácter control-L (^L).

Codificación de caracteres

Se recomienda el uso de ASCII o Latin-1 (en lugar de este último, usar UTF-8)¹

Imports

Los imports deberían colocarse en distintas líneas y siempre en la parte superior del archivo, antes de las variables globales y las constantes del módulo. Estos deben agruparse siguiendo este orden: imports de librería estándar, imports de proyectos de terceras partes relacionados e imports de aplicaciones locales/imports específicos de la librería.

Espacios en blanco en expresiones y sentencias

Se deben evitar espacios en blanco extra en las siguientes situaciones:

¹ <http://www.python.org/dev/peps/pep-3120/>

- Después de entrar o antes de salir de un paréntesis, corchete o llave.
- Antes de una coma, punto y coma o dos puntos.
- Antes de abrir un paréntesis para una lista de argumentos.
- Antes de abrir un paréntesis usado como índice o para particionar.
- No se debe dar más de un espacio alrededor de un operador de asignación (u otro operador aritmético) para alinearlos con otro.

Comentarios y documentación

Es bastante primordial que comentemos nuestro código. Imaginen que después de mucho tiempo vuelves a revisar el código que hiciste pero hay muchas cosas que no recuerdas. Es importante tener como prioridad que los comentarios se encuentren actualizados acorde a lo que se vaya desarrollando.

Los comentarios de bloque se indentan al mismo nivel que dicho código, cada línea comienza con un # y un único espacio y para el caso de los comentarios en línea que se encuentra en la misma línea de la sentencia, los comentarios deberían separarse por al menos dos espacios de la sentencia que comentan.

Para el caso de las cadenas de documentación, esta es la manera correcta de realizarlo (notar la línea en blanco antes del "" de cierre):

```
"""Documentación de ejemplo
Debe realizarse de esta manera para cumplir los estándares del PEP
"""
```

Convenciones de nombres

Para las convenciones de nombres aún no se tiene algo consistente pero sí se tiene estándares recomendados.

- **Descriptivo:**

Estos son los estilos de nombrado. Aquí encontramos a los más comunes: una letra en minúscula, una letra en mayúscula, minúsculas, mayúsculas, con guiones bajos, o considerando palabras con CamelCase, pero por ejemplo algo que debe evitarse es el uso de palabras que inicien con mayúsculas y tenga guiones bajos.

- **Prescriptivo:**

Existen 3 caracteres que deben evitarse usar como nombre de variable: la e minúscula, la o mayúscula y la i mayúscula. Los nombres de módulos deben escribirse en su totalidad con minúsculas o con guiones bajos, para el caso de las funciones también deben escribirse en minúsculas. En los argumentos de

funciones y métodos debe utilizarse siempre "self" como primer argumento de los métodos de instancia y "cls" como primer argumento de métodos de clase. El diseño que se tiene para la herencia, si no sabes bien si poner a un método o variable de instancia que sea público o no público, es mejor dejarlo como no público ya que si se necesita cambiar luego será más fácil que el hacerlo de manera inversa.

"_" interactivo

```
>>> 14 + 14
28
>>> -
28
```

Esta es realmente una característica útil que sorprendentemente poca gente conoce. En el intérprete interactivo cada vez que se evalúa una expresión o llamada a función, el resultado se almacena en un nombre temporal _ (un guión bajo). Este _ almacena la última expresión impresa. Cuando un resultado es None, nada se imprime, así que _ no cambia.

Esto solo funciona en el intérprete interactivo no dentro de un módulo

“Ahora, es mi convicción que Python es mucho más fácil que enseñar a estudiantes de programación y que enseñarles C o C++ o Java al mismo tiempo porque todos los detalles de los lenguajes son mucho más difíciles. Otros lenguajes de scripting realmente no funcionan muy bien allí. “Guido Van Rossum”

PEP8 1.4.2

Es el verificador de la guía de estilo de Python, esta herramienta permite verificar tu código con las convenciones que se tienen. La manera de instalar, actualizar o desinstalar es la siguiente:

```
$ pip install pep8
$ pip install --upgrade pep8
```

```
$ pip uninstall pep8
```

Conclusiones

Python es un lenguaje tan genial que hasta cuenta con su documento de guías de estilo. Es muy importante saber cómo escribir código limpio y legible en el lenguaje en el que desarrollamos, ya que a la larga el código que realicemos es nuestra creación. Qué mejor si esto que creamos es de ayuda para los demás y aparte de todo, si es tan claro que cumpla con todos los estándares para que trascienda con el tiempo.



APESOL

Asociación Peruana de Software Libre

<http://www.apesol.org.pe>

Tu saldo de *PayPal*

cóbralo desde cualquier parte del mundo

- ✓ Tarjeta de débito prepaga **MasterCard**
- ✓ **Compras** con tu tarjeta alrededor del mundo
- ✓ Extracción de **dinero en efectivo** desde Cajeros Automáticos
- ✓ **Cuenta bancaria virtual en USA**
(para transferir el dinero desde PayPal)

Regístrate ahora y recibe USD 25.- de regalo
con tu primera carga de USD 100.-



URL PARA REGISTRO: <http://bit.ly/payoneer-hd>

Hal 9000 Junior. (Primera Parte)

En esta oportunidad veremos como interactuar con un sensor 3D y nuestro sistema GNU/Linux. Viajaremos desde qué es un sensor 3D hasta los pasos de instalación de diferentes bibliotecas para su uso.

Escrito por: **Celia Cintas** (Licenciada en Informática)



Licenciada en Informática (UNPSJB), actualmente realizando **Doctorado Ingeniería** (Procesamiento de Imágenes, UNS), Docente (UNPSJB), Intento de sysadmin (CC) y code monkey el resto de las horas :). Pythonera por defecto con alarmantes inclinaciones hacia Prolog y otras herejías.

Webs:

Blog: <http://yetanotherlog.wordpress.com/>

Redes sociales:

Twitter / Identi.ca: [@RTFMcelia](#)

Este título (intento humorístico) se debe a que con pequeños programas podremos procesar lo que nuestro sensor observa y realizar acciones en pos de lo visto, permitiéndonos interactuar con la máquina mediante movimientos, gestos y sonidos. En esta primera parte veremos los pasos necesarios para tener nuestro sensor andando con nuestro GNU/Linux y dejaremos para la segunda parte algunos programas haciendo uso de el...

Anatomía genérica de un sensor 3D

Hablando en cuestiones de Hardware está compuesto como mínimo por:

- Sensor de Profundidad, integrado por:
 - Láser Infrarrojo.
 - Sensor CMOS monocromo.



Y dependiendo del modelo del cual dispongamos podemos encontrar agregados como:

- Motor Pivotal.
- Una cámara de tipo RGB.
- Micrófonos.

Bibliotecas Disponibles

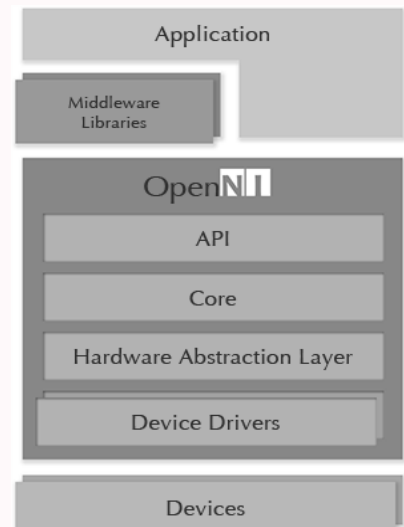
Para trabajar desde GNU/Linux, optamos por **OpenNI SDK**, que puede descargarse en <http://www.openni.org/>.

Como nos comenta OpenNI en su página, nos otorga APIs para el desarrollo de aplicaciones con sensores 3D y de audio; también en su página encontraremos *middlewares* de alto nivel para soluciones de *visual tracking*² mediante *Computer Vision*, reconstrucción de objetos en tiempo real³.

En nuestro caso veremos NITE™.

NITE™ otorga una API para el control mediante manos o cuerpo completo. Sus algoritmos utilizan profundidad, color, IR y audio como fuentes de información, dándonos detección, seguimiento y análisis de la escena, entre muchas cosas más.

Queda para futuros artículos charlar sobre *libfreenect*, un excelente proyecto.



Instalación

La instalación es *muuuuy* simple. Antes que nada debemos descargar OpenNI y NITE.

Para los *linuxeros* de 64:

```
$ wget -c http://www.openni.org/wp-content/uploads/2013/01/OpenNI-Linux-x64-2.1.0.tar.zip
```

Y los de 32:

2 <http://cvrlcode.ics.forth.gr/handtracking/>
3 http://www.artec3d.com/es/3d_scanners_for_fun/software/

```
$ wget -c http://www.openni.org/wp-content/uploads/2013/01/OpenNI-Linux-x86-2.1.0.tar.zip
```

Luego extraemos los archivos:

```
$ unzip OpenNI-Linux-x64-2.1.0.tar.zip & tar xvjf OpenNI-Linux-x64-2.1.0.tar.bz2
```

Nos ubicamos en la carpeta y nos encontraremos con el simple y querido `install.sh`, que generará las reglas en `udev` necesarias para poder utilizar los dispositivos sin ser *root*:

```
$ cd OpenNI-2.1.0-x64/  
$ sudo ./install.sh
```

Ahora haremos algo similar para instalar NITE.

En 64:

```
$ wget -c http://www.openni.org/wp-content/uploads/2012/12/NITE-Bin-Linux-x64-v1.5.2.21.tar.zip
```

Y los de 32:

```
$ wget -c http://www.openni.org/wp-content/uploads/2012/12/NITE-Bin-Linux-x86-v1.5.2.21.tar.zip
```

Se extrae de la misma forma que vimos anteriormente y corremos su `install.sh`

Ejemplos

NITE trae una batería de *scripts* en los que podemos chequear que haya quedado todo correctamente instalado, como por ejemplo:


```
$ cd Samples/Bin/x64-Release  
$ ./Sample-Players
```

En este ejemplo además de identificar usuarios se pueden encontrar esqueletos de los mismos, los que nos dejará hacer un seguimiento de sus movimientos.

```

$ ls
com.primesense.NITE.jar          libXnVnNite_1_5_2.so  Sample-CircleControl  Sample-SceneAnalysis
com.primesense.NITE.Samples.Boxes  libXnVNITE.jni.so    Sample-Players        Sample-SingleControl
com.primesense.NITE.Samples.Boxes.jar  Sample-Boxes        Sample-PointServer    Sample-TrackPad
libXnVCNITE_1_5_2.so             Sample-Boxes.net.exe  Sample-PointViewer    XnVNite.net.dll
[ 9:30 ] [ celita@tuxwoman:~/Documentos/plus/NITE-Bin-Dev-Linux-x64-v1.5.2.21/Samples/Bin/x64-Release ]
$ ./Sample-Players
509 INFO    New log started on 2013-01-26 21:31:06
592 INFO    OpenNI version is 1.5.4 (Build 0)-Linux-x86 (Jan  3 2013 18:06:45)
619 INFO    --- Filter Info --- Minimum Severity: UNKNOWN
Look for pose
Lost user 1
Look for pose
Found pose "Psi" for user 1
Calibration started
Look for pose
Calibration done [1] unsuccessfully
Lost user 1
Look for pose
Lost user 2
Found pose "Psi" for user 1
Calibration started
Calibration done [1] successfully
Lost user 1
Lost user 2
Lost user 2
Lost user 1
Lost user 1
Lost user 1

```



Teniendo todo instalado, en la segunda parte veremos pequeñas aplicaciones en las que podemos hacer uso y abuso de nuestro sensor de forma muy simple :)

Referencias y Links Interesantes

- [1] <http://www.openni.org/>
- [2] http://openkinect.org/wiki/Main_Page
- [3] <https://github.com/OpenKinect/libfreenect>
- [4] <http://depthjs.media.mit.edu/>

Scrum y eXtreme Programming

para programadores Python o PHP

Curso Online

Pair Programming - Refactoring - TDD - Planning Poker

Talleres interactivos con video en vivo

<http://cursos.eugeniabahit.com/curso-agile>

Clic aquí

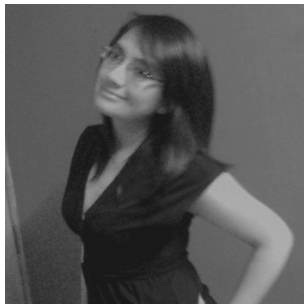


Clases individuales a cargo de
Eugenia Bahit

Pylint al rescate

Cuando nosotros desarrollamos en nuestro lenguaje preferido, en este caso Python, podríamos tomar en cuenta algunas herramientas que verifiquen si a medida que vamos avanzando, vamos bien o si ya se presentó algún error y ver como corregirlo. Pylint es una gran herramienta que viene a nuestro rescate en contra de los bugs.

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



Estudiante de Ingeniería Informática. Miembro de **APESOL** ([Asociación Peruana de Software Libre](#)) y 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 / Identi.ca: [@milale](#)

Pylint⁴ es una herramienta que verifica errores en el código de Python e intenta hacer cumplir los estándares en el código basándose en la PEP8. Esta herramienta te alerta sobre posibles fallas en el código, detecta ciertos peligros en determinado contexto y mostrando mensajes a medida que va analizando lo que “codeamos”.

Los mensajes se clasifican bajo varias categorías y al mostrar las estadísticas se compara la ejecución previa con la actual para poder ver si el código tuvo alguna mejora. Los tipos de mensaje pueden ser:

- **[R]** Refactorizar por una violación de métrica de “buena práctica”.
- **[C]** Convención por una violación al estándar de codificación.
- **[A]** Alerta para problemas de estilo o errores de programación menores.
- **[E]** Error en problemas de programación importantes (por ejemplo un *bug*).
- **[F]** Fatal por errores que impidieron su posterior procesamiento de análisis.

4 http://www.logilab.org/card/pylint_manual

Características

- **Estándar de código**

Verificar la longitud de las líneas del código, si los nombres de variable están bien formados, si los módulos importados son usados.

- **Detección de error**

Verificar si las interfaces declaradas son realmente implementadas, si los módulos están importados.

- **Ayuda de refactorización**

Pylint detecta código duplicado

- **Integración del editor e IDE**

Pylint viene integrado con varios editores e IDE⁵.

- **Diagramas UML**

Pyreverse crea diagramas UML para código de Python

- **Integración continua**

- **Extensibilidad**

Análisis estático del código en Python

El análisis estático del código se refiere al proceso de evaluación del código fuente sin ejecutarlo, la información que se obtendrá nos permitirá mejorar la línea base de nuestro proyecto. Es por esto que Pylint es muy necesario en el proceso de integración. Para algunas personas quizás los mensajes podrían resultar un poco molestos, ya que cosas que uno puede codear conscientemente y que no representan mayor relevancia para uno, para el caso de Pylint puede que sí represente algo negativo, pero todo es cuestión de considerar que mensajes deseamos que nos sean alertados y cuales no.

Ajusta la configuración de Pylint para no recibir alertas sobre ciertos tipos de advertencias o errores.

Pylint hace una serie de revisiones al código fuente:

Revisiones básicas:

- Presencia de cadenas de documentación (*docstring*)
- Nombres de módulos, clases, funciones, métodos, argumentos, variables.
- Número de argumentos, variables, retornos, métodos, etc.

⁵ http://www.logilab.org/card/pylint_manual#ide-integration

- Uso de declaraciones globales

Revisión de variables:

- Determina si una variable o import no está siendo usado.
- Variables indefinidas
- Uso de variable antes de asignación de valor.

Revisión de clases:

- Métodos sin self como primer argumento.
- Acceso único a miembros existentes vía self
- Atributos no definidos en el método `__init__`

Revisión de diseño:

- Número de métodos, atributos, variables, etc.
- Tamaño, complejidad de funciones.

Otras advertencias:

- Código fuente con caracteres no-ASCII.
- Revisión de excepciones

Ventajas

Pylint tiene una sección dedicada a reportes que se enfocan en un aspecto particular del proyecto referente a lo detallado en sus análisis y finalmente cuenta con una puntuación general basado en el número y severidad de errores y advertencias encontradas en todo el código; Pylint es totalmente configurable y permite instalar complementos *y/o plugins* para agregar funcionalidades.

*“Tengo esa esperanza, que hay un camino mejor.
Herramientas de alto nivel que actualmente te permiten ver la
estructura del software más clara serán de enorme valor.”*
Guido Van Rossum

Ejemplo

Un ejemplo del reporte que Pylint da es el siguiente:

Messages by category

type	number	previous	difference
convention	969	1721	-752.00
refactor	267	182	+85.00
warning	763	826	-63.00
error	78	291	-213.00

Es importante saber a que se refiere los código indicados por Pylint, para esto el siguiente comando es de mucha utilidad.

```
pylint --help-msg=CODIGO
```

Posibles mensajes:

- **W0232:** No existe el método `__init__`
- **R0201:** Camina como una función, habla como un método
- **W0401 & W0611 - Imports comodín**

Por ejemplo se encuentra un `from module import *`, pero este módulo importado no se está usando.

- **C0112:** Se halla un *docstring* (una cadena de documentación) vacía
- **W0301:** Punto y coma innecesario.

Y claro que no estará de más tener a la mano la página del wiki de Pylint⁶ para saber el significado exacto de cada mensaje que nos aparezca, después de ejecutar la herramienta que viene a nuestro rescate en busca de posibles *bugs*.



⁶ <http://pylint-messages.wikidot.com/all-messages>

Manual de MVC: (3) Controladores

En la primera entrega del manual, estuvimos viendo como crear el controlador principal de la aplicación. Aprendimos cómo éste, se comunicaba de forma directa con el controlador del modelo correspondiente, actuando de “ruteador” de los recursos. En esta entrega final nos concentraremos en las responsabilidades de los controladores y en los casos en las que éstas intervienen.

Escrito por: **Eugenia Bahit** (Arquitecta GLAMP & Agile Coach)



Eugenia es **Arquitecta de Software, docente** instructora de tecnologías **GLAMP** (GNU/Linux, Apache, MySQL, Python y PHP) y **Agile coach** (UTN) especializada en Scrum y eXtreme Programming. Miembro de la [Free Software Foundation](#) e integrante del equipo de [Debian Hackers](#).

Webs:

Cursos de programación a Distancia: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

Redes sociales:

Twitter / Identi.ca: [@eugeniabahit](https://twitter.com/eugeniabahit)

En la primera entrega del Manual de MVC publicada en Hackers & Developers Magazine Nro. 1, hablamos de FrontController y aprendimos que éste actúa de “ruteador” de cada recurso, instanciando directamente al controlador del modelo.

Como recordarán, cada controlador debería estar preparado -desde su método constructor- para realizar una llamada de retorno al recurso correspondiente. Dicha acción, se encontraba en relación directa a la instancia generada por FrontController.

La idea de esta última entrega en concentrarnos en los recursos que los controladores de los modelos suelen manejar, sus responsabilidades y los casos en las que las mismas intervienen.

Características de los controladores

Todo controlador llevará el nombre del modelo, seguido de la palabra Controller:

En Python

```

class VidrioController(object):
    pass

# En PHP
class VidrioController { }

```

Todo controlador debe preparar su método constructor para recibir al menos 2 parámetros en la instancia que realice el FrontController:

```

# En PHP
class VidrioController {

    function __construct($recurso='', $argumento) {
    }

}

```

En el caso de Python, además, será muy útil que el FrontController le entregue un tercer parámetro: el diccionario environ para que el controlador pueda recuperar los datos enviados desde los formularios (vía POST) a través de la clave wsgi.input (si existe). En este caso, el controlador, deberá disponer de dicho valor en una propiedad:

Por favor, notar que esto es solo necesario si se está trabajando en el supuesto de WSGI sobre Apache.

```

# En Python
class VidrioController(object):

    def __init__(self, recurso='', argumento=0, environ={}):
        self.pd = env['wsgi.input'].read() if 'wsgi.input' in environ else ''

```

Los métodos constructores deberán realizar una llamada de retorno a los recursos correspondientes, pasándoles el argumento como parámetro:

```

# En Python
class VidrioController(object):

    def __init__(self, recurso='', argumento=0, environ={}):
        self.pd = env['wsgi.input'].read() if 'wsgi.input' in environ else ''
        setattr(self, recurso)(int(argumento))

# En PHP
class VidrioController {

    function __construct($recurso='', $argumento) {
        call_user_func(array($this, $recurso), $argumento);
    }
}

```

```
}  
  
}
```

Los métodos de los controladores, representarán los recursos disponibles:

```
# En Python  
class VidrioController(object):  
  
    def __init__(self, recurso='', argumento=0, environ={}):  
        self.pd = env['wsgi.input'].read() if 'wsgi.input' in environ else ''  
        getattr(self, recurso)(int(argumento))  
  
    def agregar(self, *arg):  
        pass  
  
    def guardar(self, *arg):  
        pass  
  
    def ver(self, id=0):  
        pass  
  
# En PHP  
class VidrioController {  
  
    function __construct($recurso='', $argumento) {  
        call_user_func(array($this, $recurso), $argumento);  
    }  
  
    function agregar() { }  
  
    function guardar() { }  
  
    function ver($id=0) { }  
  
}
```

Básicamente existen 5 tipos de recursos estándar:

agregar()	Muestra el formulario para crear un nuevo objeto
editar(id)	Muestra el formulario para modificar un objeto existente
guardar()	Guarda los cambios realizados a través de editar() y agregar()
eliminar(id)	Elimina un objeto existente
listar()	(opcional) Muestra la colección completa de objetos basados en el mismo modelo
ver(id)	(opcional) Permite ver los datos de un objeto

Los recursos editar(), eliminar() y ver() suelen recibir la ID del objeto como parámetro (argumento recibido desde FrontController).

Responsabilidades de los controladores

1. Llamar al modelo (crear instancia). Generalmente en los recursos editar, guardar, eliminar y listar (en este último caso, llamará al colector).
2. Sanear y asegurar los datos recibidos desde el usuario ANTES de modificar el modelo: el saneamiento y aseguración de datos recibidos por el usuario, está en manos de los controladores y NO de los modelos.
3. Modificar sus propiedades cuando sea necesario. Generalmente en los recursos donde realiza llamadas al modelo.
4. Hacer las llamadas (al) a los métodos correspondientes del modelo. De igual forma que los anteriores, en los casos en los que se llama al modelo y se lo modifica.
5. Entregar los datos obtenidos a la vista:
 - En el recurso agregar, suele llamar directamente a la vista sin pasar datos;
 - En los recursos editar, ver y listar, entrega los datos del objeto a la vista;
 - En los recursos eliminar y guardar, suele pasar de largo a la vista y en cambio, redirige al usuario hacia otro recurso.

¿Cómo se ven los métodos de un controlador?

Un ejemplo con código comentado:

```
# En Python
class VidrioController(object):

    def __init__(self, recurso='', argumento=0, environ={}):
        self.pd = env['wsgi.input'].read() if 'wsgi.input' in environ else ''
        getattr(self, recurso)(int(argumento))

    def agregar(self, *arg):
        # Llama directamente a la vista
        vista = VidrioView()
        self.output = vista.mostrar_form_alta()

    def editar(self, id=0):
        # Instancia al modelo
        modelo = Vidrio()
        # Modifica las propiedades necesarias
        modelo.vidrio_id = int(id)
        # Llama al método correspondiente (necesita recuperar el objeto)
        modelo.get()
        # Le entrega la información a la vista
        vista = VidrioView()
        self.output = vista.mostrar_form_edicion(modelo)

    def guardar(self, *arg):
```

```

# Obtiene los datos enviados del el formulario
# Esto generalmente debe hacerse desde un Helper
post = self.pd.split('&')
_POST = {}
for par in post:
    field, value = par.split('=')
    _POST[field] = value

# Instancia al modelo
modelo = Vidrio()
# Modifica las propiedades necesarias
modelo.vidrio_id = int(_POST['vidrio_id'] if 'vidrio_id' in _POST else 0)
modelo.color = _POST['color']
# Llama al método correspondiente (necesita guardar el objeto)
modelo.save()
# Pasa de largo a la vista y en cambio, recurre a otro recurso
self.listar()

def eliminar(self, id=0):
    # Instancia al modelo
    modelo = Vidrio()
    # Modifica las propiedades necesarias
    modelo.vidrio_id = int(id)
    # Llama al método correspondiente (necesita destruir al objeto)
    modelo.destroy()
    # Pasa de largo a la vista y en cambio, recurre a otro recurso
    self.listar()

def listar(self, *arg):
    # Recurre al colector para traer toda la colección de objetos Vidrio
    coleccion = VidrioCollector().get()
    # Le entrega la información a la vista
    vista = VidrioView()
    self.output = vista.mostrar_listado(coleccion)

```

```

# En PHP
class VidrioController {

    function __construct($recurso='', $argumento=0) {
        call_user_func(array($this, $recurso), $argumento);
    }

    function agregar() {
        # Llama directamente a la vista
        $vista = new VidrioView();
        $vista->mostrar_form_alta();
    }

    function editar($id=0) {
        # Instancia al modelo
        $modelo = new Vidrio();
        # Modifica las propiedades necesarias
        $modelo->vidrio_id = (int)$id;
        # Llama al método correspondiente (necesita recuperar el objeto)
        $modelo->get();
        # Le entrega la información a la vista
        $vista = new VidrioView();
        $vista->mostrar_form_edicion($modelo);
    }
}

```

```

function guardar() {
    # Instancia al modelo
    $modelo = Vidrio()
    # Modifica las propiedades necesarias
    $id = isset($_POST['vidrio_id']) ? (int)$_POST['vidrio_id'] : 0;
    $modelo->vidrio_id = $id;
    $modelo->color = $_POST['color'];
    # Llama al método correspondiente (necesita guardar el objeto)
    $modelo->save();
    # Pasa de largo a la vista y en cambio, recurre a otro recurso
    $this->listar();
}

function eliminar($id=0) {
    # Instancia al modelo
    $modelo = new Vidrio();
    # Modifica las propiedades necesarias
    $modelo->vidrio_id = (int)$id;
    # Llama al método correspondiente (necesita destruir al objeto)
    $modelo->destroy();
    # Pasa de largo a la vista y en cambio, recurre a otro recurso
    $this->listar();
}

function listar() {
    # Recurre al colector para traer toda la colección de objetos Vidrio
    $coleccion = VidrioCollector::get();
    # Le entrega la información a la vista
    $vista = new VidrioView();
    $vista->mostrar_listado($coleccion);
}
}

```

Curso de Arquitecturas MVC con Python o PHP

8 semanas · Clases individuales

Chat Telefónico + Pantalla compartida + E-mail

Tutoría personalizada a distancia (online)

<http://cursos.eugeniabahit.com/curso-mvc>

Clic aquí



Clases individuales a cargo de
Eugenia Bahit

Double Test con ZendFramework2

A veces se desean hacer pruebas a un *system under test* (SUT) que depende de uno o mas componentes, los cuales no pueden ser incorporados al entorno de pruebas. Debido a esto nacen los *Doubles Test*, que separan el objeto que se está probando de otros objetos (de los cuáles se depende pero no se necesita probar), disminuyendo sus dependencias. De esta manera se mantienen las pruebas de integración como pruebas unitarias. Hablaremos un poco mas de su función y de cómo se implementan.

Escrito por: **Indira Burga** (Ingeniera de Sistemas)



Indira es **Ing. de Sistemas** de Perú. Gestora de Proyectos de desarrollo de software, **programadora PHP**, analista, nueva amante de las **metodologías Ágiles**. Ahora envuelta en una nueva aventura: su propia empresa "**IC Projects**" dedicada al desarrollo de Software.

Webs:

About.me: <http://about.me/indirabm>

Redes sociales:

Twitter: [@indirabm](https://twitter.com/indirabm)

Como teoría básica para toda prueba unitaria es que el SUT se encuentre aislado del sistema debido a que no se desea que éste ejecute métodos que puedan modificar la data. Al existir dependencias de éstos se utilizan los *Double Test* que reemplazan las dependencias dándonos las respuestas que deseamos para que la prueba se ejecute correctamente.

También suele utilizarse cuando tenemos algún error en producción, en un método que no ha sido testeado. Para este caso, lo ultimo que deseamos es que se ejecuten los otros métodos dependientes así que hacer un test doble es la respuesta.

“Cuando nosotros escribimos una prueba en la cual podríamos (o no) usar una real dependencia de componentes

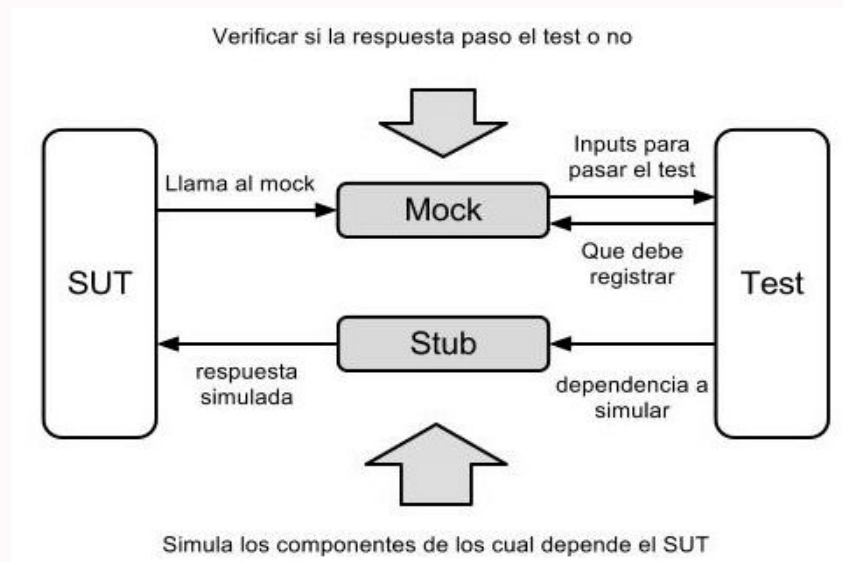
(DOC), nosotros podemos reemplazarlo con un Doble de Pruebas. El doble de pruebas no tiene que comportarse exactamente como el real DOC; simplemente tiene que proporcionar la misma API que la real de modo que el SUT piensa que es el verdadero!” – Gerard Meszaros

Dentro de los *doubles tests*, se suele hablar de los Stubs y Mocks, aunque también están los Spies, Fakes y Dummy. Sin embargo aquí hablaremos de los dos primeros.

Stubs: Reemplaza el componente del que depende el SUT y este test sirve como punto de control para los *inputs* indirectos del SUT.

Mocks: Objeto pre-programado con expectativas que conforma la especificación de cómo se espera que se reciban las llamadas

Pero como en realidad funciona, aquí les dejo un gráfico que muestra a groso modo la función de los stub y mock en una prueba.



A continuación mostraremos dos ejemplos: uno de stubs y otro de mocks, en Zend Framework 2.

Para esto necesitas contar con un proyecto en Zend Framework 2; si no lo tienes sigue estos pasos:

- Descargar Zend Framework 2. Este manual⁷ es muy bueno, sin embargo cuando llegas a *Unit Testing* no funciona.
- Para Unit Testing, usar este tutorial⁸.
- Ahora necesitamos un proyecto con el cual realizar las pruebas, por suerte nuestros amigos de ZF nos la han dejado fácil y cuentan con un ejemplo⁹

7 <http://zf2.readthedocs.org/en/latest/user-guide/skeleton-application.html>

8 <http://framework.zend.com/manual/2.0/en/user-guide/unit-testing.html>

9 <http://zf2.readthedocs.org/en/latest/user-guide/modules.html>

muy bueno.

Es hora de los ejemplos:

Stub en ZF2

En Album\Controller\AlbumController.php crear el siguiente método.

```
public function doSomethingAction($name='Nada')
{
    // Do something.
    return $name;
}
```

Stub nos permite especificar una instancia de la clase AlbumController la cual va a ser codificada para retornar un valor conocido al método doSomethingAction, para que cuando se hagan pruebas y se continúe desarrollando en este método, siempre se tenga una respuesta conocida para los Test.

En test/controller/AlbumControllerTest.php crear la siguiente función.

```
public function testStub()
{
    /**
     * Crear un Stub para el controlador Album.
     * En lugar de llamar a una clase estática se llama al método getMock()
     */
    $stub = $this->getMock('Album\Controller\AlbumController',
                        array('doSomethingAction'));

    // Configurar el stub.
    $stub->expects($this->any())
        ->method('doSomethingAction')
        ->will($this->returnArgument(0));

    /**
     * Llamando al método $stub->doSomething() que devolverá
     * 'algo'.
     */
    $this->assertEquals('algo', $stub->doSomethingAction('algo'));
}
```

Problemas encontrados:

\$this->getMock()

Al identificar o definir correctamente la clase de Zend para que funcione getMock con ZF2, se debe de especificar la clase como Album\Controller\AlbumController, y no como AlbumController.

->method()

En un método en ZF siempre debe existir el Action al final del nombre del

método, pero cuando se usa, se obvia la palabra Action. Para el Caso de method('nombreDeMetodo') se debe indicar el nombre real nombreDeMetodoAction, porque es ahí donde se tiene error con ZF2.

Mock en ZF2

Vamos a probar que la clase AlbumTable interactúe con la clase TableGateway de la forma adecuada. Cabe señalar que la clase TableGateway está mas que comprobado su funcionamiento, pero solo es por cuestiones de prueba.

```
public function testFetchAllReturnsAllAlbums()
{
    // Array de Datos en Fila (vacío)
    $resultSet      = new ResultSet();

    /**
     * Crear un mock para la clase Zend_Db_TableGateway que es un objeto que
     * representa a una tabla en la Base de Datos.
     * Para ellos solo vamos a usar el método "select"
     */
    $mockTableGateway = $this->getMock('Zend\Db\TableGateway\TableGateway',
                                     array('select'), array(), '', false);

    //configurar Mock con el metodo "Select"
    $mockTableGateway->expects($this->once())
        ->method('select')
        ->with()

    // retorna valore de Array de Datos en Fila ( vacio )
    ->will($this->returnValue($resultSet));

    /**
     * Llamamos a la clase AlbumTable porque queremos probar que la Clase
     * AlbumTable tiene un método que usa la clase TableGateway, por eso
     * le pasamos todo la respuesta de $mockTableGateway
     */
    $albumTable = new AlbumTable($mockTableGateway);

    /**
     * Aserción
     * Aquí debemos de observar que la clase AlbumTable tiene un método
     * que se llama "fetchAll()" el cual lo único que hace es llamar
     * a $this->tableGateway->select() Lo cual lo convierte en una
     * prueba de Doble test
     */
    $this->assertSame($resultSet, $albumTable->fetchAll());
}
```

Finalmente tenemos que tener cuidado al utilizar *double test* porque estamos probando el SUT en una configuración diferente de la que se utilizará en producción. Así que realmente debemos tener por lo menos una prueba que verifica que funciona realmente sin el *double test*.

¿Cómo crear aplicaciones Web PHP con EuropioEngine?

EuropioEngine es un motor para aplicaciones Web modulares desarrolladas en PHP y que requieran de bases de datos MySQL para persistencia de objetos. A diferencia de un Framework, EuropioEngine -al ser un motor de Software y no un Framework-, no requiere aprender una sintaxis particular ni nada parecido: solo necesitas programar en lenguaje PHP puro.

Escrito por: **Eugenia Bahit** (Arquitecta GLAMP & Agile Coach)



Eugenia es **Arquitecta de Software**, **docente** instructora de tecnologías **GLAMP** (GNU/Linux, Apache, MySQL, Python y PHP) y **Agile coach** (UTN) especializada en Scrum y eXtreme Programming. Miembro de la **Free Software Foundation** e integrante del equipo de **Debian Hackers**.

Webs:

Cursos de programación a Distancia: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

Redes sociales:

Twitter / Identi.ca: [@eugeniabahit](https://twitter.com/eugeniabahit)

A mediados del año 2006, comencé a desarrollar -sin pensarlo ni pretenderlo-, un Framework en PHP que -al igual que cualquier Framework desconocido por mi en aquel momento-, por un lado, le evitara a los programadores de mi equipo el desarrollo de funcionalidades y/o tareas de programación repetitivas y por otro, nos permitiera acelerar los procesos de desarrollo: pues la economía estaba atravesando una crisis y nos veíamos obligados a asumir mayor cantidad de proyectos de los que realmente podíamos afrontar.

Así surge Europio en sus inicios, pero bajo el nombre de ZK-PHPFramework. Años después y vendida la empresa que lideraba por aquel entonces, ZK-PHPFramework quedó en el olvido, resurgiendo recién a fines de 2010 como Europio: ya no tenía aspecto de Framework y comenzaba a parecerse cada vez más a un *Software Engine* (motor de Software).

A fines de 2012, me propuse terminar de convertirlo en un **verdadero motor de Software**, ya que venía trabajando “sutilmente” en ello, desde el mes de julio. El pasado 7 de febrero de 2013, el proyecto se hizo realidad: la versión 3.0 alfa 73 estaba disponible para solo unos días después, lanzar varias versiones hasta llegar a la **versión 3.0 beta 2 de EuropioEngine**, en la que nos enfocaremos en este artículo.

Las características completas de EuropioEngine así como el manual de uso y su documentación, puedes obtenerlo ingresando en el sitio Web oficial, www.europio.org. La idea ahora, es “poner en marcha el motor” para entender como se puede desarrollar una aplicación implementando este motor.

EuropioEngine ha sido pensado para desarrollar Software de alta envergadura y no sirve para desarrollar sitios Web ni blogs.

Instalación y Configuración

Lo primero que tenemos que hacer es descargar la versión 3.0 beta 2 de EuropioEngine:

```
~$ wget http://www.europio.org/downloads/EuropioEngine_3_0_beta_2.tar.gz
```

Una vez descargado, vamos a descomprimirlo:

```
~$ tar -xzvf EuropioEngine_3_0_beta_2.tar.gz
```

Ahora, vamos a crear la base de datos que utilizará nuestra aplicación:

```
~$ mysql -u root -p
mysql> CREATE DATABASE europioengine; \q
```

Una vez creada la base de datos, vamos a recurrir al **CLI de EuropioEngine** para crear la tabla de usuarios y configurar al administrador general del sistema:

```
~$ cd EuropioEngine/core/cli
~$ ./europio --user-setup europioengine
```

user-setup nos pedirá ingresar:

- Nombre de usuario para el nuevo administrador
- Contraseña para ese usuario
- Finalmente, MySQL nos pedirá la clave del usuario root

EuropioEngine CLI es un conjunto de *Shell Scripts* que pueden localizarse en el directorio `EuropioEngine/core/cli`

Ahora, tendremos que realizar las configuraciones básicas para que nuestra aplicación, funcione de forma correcta. Para ello, el primer paso es copiar el archivo `settings.php.dist` como `settings.php` (**no recomiendo renombrarlo**, sino conservar una copia del archivo de distribución):

```
~/EuropioEngine/core/cli$ cd ../../
~/EuropioEngine$ cp settings.php.dist settings.php
```

Abrimos el nuevo archivo `settings.php` y modificamos el valor de las siguientes constantes:

```
const DB_HOST = "localhost";
const DB_USER = "root";
const DB_NAME = "europioengine";

# Aquí, escribe la clave del root de MySQL o del usuario MySQL que hayas indicado
const DB_PASS = "pwd";

# Asegúrate de indicar el path absoluto hasta la carpeta EuropioEngine
# Notar que ambas rutas DEBEN finalizar con una /
const APP_DIR = "/home/tu-usuario/EuropioEngine/";
const STATIC_DIR = "/home/tu-usuario/EuropioEngine/static/";

const DEFAULT_VIEW = "/users/user/listar";
```

Guardamos los cambios y solo nos resta configurar un VirtualHost para nuestra aplicación. Primero, vamos a crear una carpeta para guardar los *logs* de Apache para esta aplicación y que nos sea mucho más simple conocer los errores que, eventualmente, pueda generar:

```
~/EuropioEngine$ mkdir ../logs
```

Y ahora sí, creamos nuestro Virtual Host:

```
~# cd /etc/apache2/sites-available
# notar que la ruta anterior puede cambiar de acuerdo a cada distribución

/etc/apache2/sites-available# touch europioengine.local
```

Abrimos el archivo y copiamos el siguiente código:

```
<VirtualHost *:80>
  ServerName europioengine.local

  # Modificar tu-usuario por el indicado (o toda la ruta según el caso)
  DocumentRoot /home/tu-usuario/EuropioEngine
  ErrorLog /home/tu-usuario/logs/europio_errors.log
  CustomLog /home/tu-usuario/logs/europio_access.log combined
  <Directory />
    Options -Indexes
    AllowOverride All
  </Directory>
</VirtualHost>
```

Guardamos el archivo y habilitamos el VirtualHost:

```
/etc/apache2/sites-available# a2ensite europioengine.local
```

Reiniciamos Apache:

```
/etc/apache2/sites-available# service apache2 restart
```

Agregamos el *host*:

```
/etc/apache2/sites-available# echo "127.0.0.1 europioengine.local" >> /etc/hosts
```

EuropioEngine (a partir de la versión 3.0 beta 1), trae incorporado un módulo de usuarios (un sencillo ABM que podría reemplazarse si así se desea). Si quieres, puedes probar este módulo indicando la siguiente URL en tu navegador Web:

```
http://europioengine.local/
```

Te enviará directamente al formulario para iniciar sesión:



Usuario y Contraseña

Usuario desconectado del sistema

admin

.....

Ingresar

[Página principal](#)

Para **modificar los estilos del formulario** edita el archivo `EuropioEngine/static/css/login.css`. Sino, para **crear tu propio *template***, reemplaza el archivo `EuropioEngine/static/html/login.html`. Estos archivos estáticos, están allí, solo a modo de ejemplo a fin de que utilizar `EuropioEngine` no sea demasiado complicado.

Creando nuestro primer módulo

En principio se debe tener presente que `EuropioEngine` maneja **arquitecturas MVC estrictas**. Esto significa que no existe forma de poder utilizar el motor si la arquitectura de tu aplicación y estructura de directorios, no es la correcta. Pues el *roteador* de `EuropioEngine` te dirigirá indefectiblemente a la vista por defecto (especificada en la constante `DEFAULT_VIEW` del `settings`, cuando no pueda encontrar el recurso solicitado.

Entonces, todos los módulos de tu aplicación, deben almacenarse sí o sí, en la carpeta **appmodules** de `EuropioEngine`.

Necesitarás cumplir con una arquitectura MVC estricta:

- 1 carpeta por módulo
- Cada módulo, con 3 subdirectorios: `models`, `views` y `controllers`

Por ejemplo:

```
├─ appmodules
  └─ contabilidad
     ├── controllers
     └── models
```

```
└─ views
```

La forma más sencilla de crear un nuevo módulo, es a través del CLI de EuropioEngine:

```
~/EuropioEngine/core/cli$ ./europio -c nombredelmodulo
```

Por ejemplo:

```
~/EuropioEngine/core/cli$ ./europio -c contabilidad
```

...creará la estructura de directorios mostrada anteriormente.

Un módulo es aquella porción de un sistema informático que de forma individual, podría considerarse “una aplicación independiente con posibilidad de ser implementada en diversos programas sin requerir modificaciones”.

En MVC, un módulo (aplicación interna del programa) se encuentra dividido en 3 secciones: modelos, vistas y controladores.

Un modelo, agrupará aquellas clases (definiciones de objetos) que pertenezcan a un mismo tipo. Por ejemplo, si se tuviesen los objetos Usuario más UsuarioAdministrador heredado del primero, ambas clases se agruparían en el modelo principal llamado usuario. Es decir, que **todas las posibles definiciones para un mismo tipo de objeto, se agrupan en un modelo**. Aunque a veces, esta regla debe ser rota a fin de salvaguardar la mejor resolución de requerimientos.

A la vez, cada modelo, tendrá su controlador correspondiente, sus GUI y la lógica para procesarla (la vista).

EuropioEngine utiliza dicho patrón arquitectónico (MVC) de forma estricta. Se debe ser consciente de que MVC es un patrón arquitectónico y no, uno de diseño. Tener bien presente esta radical diferencia, facilitará la implementación de **EuropioEngine** en tu aplicación.

Replicar dicho patrón dentro de cada módulo es sumamente sencillo. Se puede utilizar **EuropioEngine CLI** para **crear los archivos y estructura de cada modelo, vista y controlador**:


```
~/EuropioEngine/core/cli$ ./europio -f contabilidad Factura
```

El CLI de EuropioEngine producirá los siguientes archivos:

```
contabilidad
├── controllers
│   └── factura.php
├── models
│   └── factura.php
└── views
    └── factura.php
```

Si observamos el código del modelo Factura, podremos observar lo siguiente:

```
class Factura extends StandardObject {
    public function __construct() {
        $this->factura_id = 0;
        # aquí se definen el resto de las propiedades
    }
}
```

Por defecto, todo objeto heredará de **StandardObject** quien otorgará a cada uno de sus “hijos”, tres métodos estándar, a saber:

```
save()      Guarda un nuevo objeto u objeto existente
get()       Recupera un objeto existente si la ID existe
destroy()  Destruye un objeto existente si la ID existe
```

Es decir, que con respecto a los modelos, lo mínimo necesario será:

1. Terminar de definir sus propiedades;
2. Mapear el modelo y crear su tabla correspondiente.

Definiendo las propiedades:

```
class Factura extends StandardObject {
    public function __construct() {
        $this->factura_id = 0;
        $this->fecha = '';
        $this->numeracion = 0;
    }
}
```

```
}
```

Mapeando el modelo y creando su tabla:

```
~/EuropioEngine/core/cli$ mysql -u root -p
mysql> use europioengine;
mysql> CREATE TABLE factura (
mysql> factura_id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
mysql> fecha DATE NOT NULL,
mysql> numeracion INT(8) NOT NULL
mysql> ) ENGINE=InnoDB;
mysql> \q
```

Contar con un ORM que provea a los modelos una independencia absoluta con respecto a la base de datos para poder manipular un objeto persistente, es una decisión acertada puesto que permite poder programar una aplicación 100% orientada a objetos, manipulando solo “objetos” en vez de manipular datos. Esto hace a la aplicación mucho más fácil de mantener y evolucionar, ya que nunca estará atada al diseño de una base de datos.

Sin embargo, utilizar un ORM para mapear un objeto que no es genérico a fin de crear su estructura de datos, es tan poco prolijo como inconveniente: pues un ORM no tiene la capacidad humana para optimizar una estructura de datos siendo que indefectiblemente se necesitará la intervención humana. Ya sea ésta, para realizar un mapeo manual como para para indicar dentro del constructor (o en un método diseñado al fin) las características y requerimientos de cada propiedad, ensuciando así los modelos mediante el agregado de *anti-patron*es en la definición de atributos.

*Los objetos no genéricos, siempre **DEBEN** ser mapeados por un humano, para garantizar: a) un diseño óptimo de su estructura de datos; b) un diseño de objetos puros, libres de anti-patron*es.

En pro de automatizar los procesos y tareas de programación, la mayoría de los ORM y de los *Frameworks* disponibles en el mercado, obligan a definir las propiedades de un objeto, ensuciando los métodos constructores o las clases con el agregado de anti-

patrones imponiendo, de esta forma, un concepto equivocado de la programación orientada a objetos. Pero todos estos conceptos son tenidos en cuenta por EuropioEngine.

EuropioEngine ha sido concebido bajo la premisa de que a aquellos programadores que lo implementen, les apasiona programar y no buscan “codear” menos, sino por el contrario, solo buscan optimizar sus aplicaciones para hacerlas más robustas, fáciles de mantener y de evolucionar.

Por lo tanto, una vez disponible la estructura de datos correspondiente, dispondrás de métodos `get()`, `save()` y `destroy()` heredados y sin necesidad de definirlos, para todos tus objetos. Sin embargo, cuando se trate de objetos no genéricos como es el caso de aquellos heredados de `StandardObject`, deberás diseñar su estructura de datos de forma manual. Con el resto de objetos genéricos, podrás utilizar el CLI para crearlas mediante el ORM.

Para crear la estructura de datos de objetos heredados de `StandardObject`, puedes recurrir al **capítulo VII** del libro **Teoría sintáctico-gramatical de Objetos**, donde explico detalladamente cómo realizar el diseño de una estructura de datos para cada objeto al que se desee hacer persistir en el sistema. Puedes obtener una **copia impresa del libro** o su **versión digital gratuita**, ingresando en:

<http://www.bubok.es/libros/219288>

En el mismo libro, también podrás encontrar información que te ayude a comprender mejor, los **objetos de los que dispone EuropioEngine**:

- `StandardObject` (objeto estándar)
- `SerializedObject` (objeto estándar “frozen”)
- `LogicalConnector` (conector lógico relacional)
- `MultiplierObject` (objeto relacional simple -o multiplicador-)
- `CollectorObject` (objeto colector)

Completando las vistas y los controladores

Los controladores creados por el CLI de EuropioEngine, poseen predefinidos 5 recursos estándar:

agregar()	Muestra la GUI del formulario para crear un nuevo objeto <i>llama directamente a la vista</i>
editar()	Muestra la GUI del formulario para modificar un objeto existente <i>Primero recurre al método get() del objeto</i> <i>Luego, le entrega dicha información a la vista</i>
guardar()	Guarda un nuevo objeto creado desde agregar o modificado desde editar <i>Modifica las propiedades del objeto (requiere que se complete)</i> <i>Recurre al método save() del objeto</i> <i>Redirige al usuario al recurso listar</i>
listar()	Muestra la colección de todos los objetos creados <i>Recurre al objeto genérico CollectorObject</i> <i>Le entrega la colección de objetos a la vista</i>
eliminar()	Destruye un objeto <i>Recurre al método destroy() del objeto</i> <i>Redirige al usuario al recurso listar</i>

El único recurso del controlador que en principio, necesitará de intervención humana para completar aquellas propiedades que serán modificadas, será **guardar()**:

```
public function guardar() {
    $id = (isset($_POST['id'])) ? $_POST['id'] : 0;
    $this->model->factura_id = $id;
    # se deben modificar todas las propiedades del modelo
    $this->model->save();
    HTTPHelper::go("/contabilidad/factura/listar");
}
```

Un ejemplo, podría ser el siguiente:

```
public function guardar() {
    $id = (isset($_POST['id'])) ? $_POST['id'] : 0;
    $this->model->factura_id = $id;
    $this->model->fecha = $_POST['fecha'];
    $this->model->numeracion = $_POST['numeracion'];
    $this->model->save();
    HTTPHelper::go("/contabilidad/factura/listar");
}
```

Por favor, notar que cualquier saneamiento o acción para asegurar los datos recibidos desde un formulario o URI, deberá ser realiza en el controlador, ya sea de forma directa o mediante un Helper.

Los métodos de la vista, han sido definidos por defecto mediante el CLI de EuropioEngine:

```

class FacturaView {

    public function agregar() {
        $str = file_get_contents(
            STATIC_DIR . "html/contabilidad/factura_agregar.html");
        print Template('Agregar Factura')->show($str);
    }

    public function editar($obj=array()) {
        $str = file_get_contents(
            STATIC_DIR . "html/contabilidad/factura_editar.html");
        $html = Template($str)->render($obj);
        print Template('Editar Factura')->show($html);
    }

    public function listar($coleccion=array()) {
        $str = file_get_contents(
            STATIC_DIR . "html/contabilidad/factura_listar.html");
        $html = Template($str)->render_regex('LISTADO', $coleccion);
        print Template('Listado de Factura')->show($html);
    }
}

```

Por defecto, EuropioEngine ha definido 3 rutas para **3 archivos HTML que deben ser creados**. Podrás crear estos archivos directamente en la ruta sugerida por **EuropioEngine** o modificar dichas rutas y crear los archivos en un directorio diferente.

Una vez completados los archivos HTML, ya tendrás tu aplicación lista para ser probada:

```
http://europioengine.local/contabilidad/factura/listar
```

Habilitando la API

Es posible habilitar un servicio web **REST/JSON** para los recursos públicos de cualquiera de tus modelos. Habilitaremos el recurso listar para el modelo Factura.

Abre el archivo `./EuropioEngine/settings.php` y busca el *array* `$allowed_resources`. Agrega una clave llamada `factura` cuyo valor sea un *array* con el elemento `listar`:

```

$allowed_resources = array(
    "factura" => array('listar'),
);

```

Para habilitar recursos públicos en la API, solo debes agregar al array, una clave con el nombre del modelo (en minúsculas) y el valor asociado, será un array simple, donde

cada elemento de éste, será el nombre de los recursos públicos que se deseen habilitar.

Una vez habilitado el recurso, se deberá indicar al controlador de dicho cambio. Todo controlador heredado de Controller, posee dos propiedades: \$api y \$apidata. La primera retorna True si el recurso está siendo llamado por la API y la segunda, con valor NULL por defecto, almacenará el o los objetos solicitados para ser codificados en formato JSON y retornados por el servidor al cliente de la API.

Un recurso con API habilitada, antes de entregar los datos a la vista deberá verificar no estar siendo llamado por la API. Si no es llamado por la API, entrega los datos a la vista. Caso contrario, se limita a almacenar dichos datos en la propiedad \$apidata:

```
public function listar() {
    $collection = CollectorObject::get('Factura');
    $list = $collection->collection;
    if(!$this->api) {
        $this->view->listar($list);
    } else {
        $this->apidata = $list;
    }
}
```

Con solo ese pequeño cambio, **tu API pública ya se encuentra habilitada y dispones de un Web Service REST**. La URL de tu API pública siempre será la misma que la de tu recurso, pero anteponiendo api/ al nombre del módulo:

```
http://europioengine.local/api/contabilidad/factura/listar
```

EuropioEngine para Hackers

Entender el funcionamiento interno de EuropioEngine ayudará a realizar las modificaciones necesarias al núcleo del motor, adaptarlo a requerimientos específicos o simplemente, a optimizarlo, mejorarlo y/o a hacerlo más robusto y potente.

Sobre el proceso de arranque del motor:

1. La aplicación arranca con la petición HTTP del usuario, estando en manos de Apache a través del archivo .htaccess localizado en la raíz.
2. Todas las peticiones se reciben en app_engine.php (excepto aquellas a archivos estáticos dentro del directorio static) quien realiza lo siguiente:
 - importa los archivos principales del núcleo (aquellos que agrupan las

- acciones para llevar a cabo una determinada tarea)
- enciende el motor MVC (`FrontController::start()`)
3. Quien realmente hará "arrancar" al motor MVC, es el `FrontController` (`core/mvc_engine/front_controller.php`) que recurrirá a `ApplicationHandler` (`mvc_engine/apphandler.php`) para analizar la URI;
 4. `ApplicationHandler` obtiene el nombre del módulo, del modelo y del recurso; además, comprueba si dicho recurso está siendo solicitado o no por la API y si se ha pasado un argumento en la URI, lo releva. Todos estos datos son recuperados por `FrontController`, quien se encargará de suministrar dicha información al controlador propietario del recurso solicitado.
 5. Aquí, comienza a actuar el controlador de usuario (es decir, el controlador creado por el usuario para un modelo determinado). Dicho controlador -que solo define recursos si hereda de `Controller` (`mvc_engine/controller.php`) -caso contrario deberá definir un constructor siguiendo como modelo el constructor de `Controller` se activa de forma automática a través del constructor heredado de éste.
 6. El constructor heredado de `Controller`, verifica que el recurso exista y pueda ser llamado (caso contrario, redirige a la vista por defecto).
 7. Luego, realiza previamente una instancia del modelo y otra de la vista y realiza una llamada de retorno al recurso solicitado. A partir de allí, interviene el recurso de usuario (es decir, método definido por el usuario en el controlador de usuario).

Sobre la automatización de modelos y los factores de herencia real:

La automatización de modelos es en realidad una relación de herencia real: todos los objetos, sean cuáles sean, son subtipos de un objeto. `EuropioEngine` no considera el concepto de "modelo genérico", "Model" o "DataModel", sino que cumple con una orientación a objetos absoluta y sumamente estricta ya que no analiza modelos orientados a datos, sino objetos relacionados mediante un único contenedor. Un objeto antes de ser "ObjetoX" es primeramente "Objeto". Y éste, se encuentra representado por `StandardObject` (`core/orm_engine/objects/standardobject.php`).

`StandardObject` -como tal- es abstracto (solo puede heredarse de él ya que en sí mismo, no cuenta con una definición precisa. No es más que la abstracción de un objeto cualquiera).

Define los 3 métodos estándar `save()`, `get()` y `destroy()`. Los mismos, utilizan los métodos CRUD de un ORM para comunicarse con la base de datos.

El ORM (`orm_engine/orm/orm.php`) se ayuda de un *Helper* (`orm_engine/orm/helper.php`) para crear los *queries* y obtener otros datos necesarios para ejecutar las consultas. El *Helper* obtiene toda la información necesaria para la creación de *queries* y configuración de datos, del análisis realizado por el *ORMHandler* (`orm_engine/orm/handler.php`).

Finalmente, el ORM se comunicará con la base de datos mediante una capa de abstracción que utiliza la API MySQLi (`orm_engine/mysqlilayer.php`) explicada en el ejemplar Nro. 0 de este mismo *Magazine*.

Además de `StandardObject` se provee otra abstracción: `SerializedObject` (de nivel experimental). Su funcionamiento es exacto al de `StandardObject` pero con él, se busca "congelar" un objeto haciéndolo inmutable frente a cambios en las relaciones de dependencia. Esto significa que *a nivel objeto es relacional pero a nivel datos no lo es*.

`SerializedObject` (`orm_engine/objects/serializedobject.php`) tiene su propio ORM (`orm_engine/sorm/sorm.php`) el cual trabaja de forma similar al ORM de `StandardObject` pero prescinde de un manejador (*Handler*) ya que dichos objetos no requieren de un análisis, sino que son directamente congelados y almacenados.

Otros tres objetos genéricos (no abstractos) se encuentran disponibles:

`CollectorObject` (`orm_engine/objects/collectorobject.php`):

un *singleton* colector que de forma iterativa recupera todos los objetos creados pertenecientes a un mismo contenedor. No requiere de un ORM y solo se ayuda de un *Helper* definido en el mismo archivo.

`MultiplierObject` (`orm_engine/objects/multiplierobject.php`):

se encarga de crear relaciones entre *1 objeto (A) y N objetos (B) idénticos* (misma ID). También cuenta con su propio ORM (`orm_engine/morm/morm.php`) quien sigue los mismos procesos que el ORM de `StandardObject`.

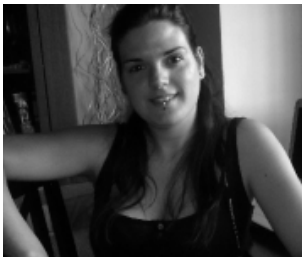
`LogicalConnector` (`orm_engine/objects/logicalconector.php`):

un conector lógico relacional que se encarga de crear relaciones entre *1 objeto (A) y N objetos (B) de identidad única* (distinta ID). También cuenta con su propio ORM (`orm_engine/corm/corm.php`) quien sigue los mismos procesos que los anteriores.

Manual de Perl (Parte II)

En el artículo anterior vimos una introducción a Perl. Ahora profundizaremos un poco más con los operadores y las estructuras de control.

Escrito por: **María José Montes Díaz** (Archera & 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://archninfablogspot.com.es/>

Redes sociales:

Twitter: [@MMontesDiaz](https://twitter.com/MMontesDiaz)

Es importante que tengamos instalado el intérprete que normalmente se incluye en todas las distribuciones y se instala por defecto, pero si por algún motivo no lo tenemos instalado podemos encontrar todas las versiones disponibles para todos los sistemas en www.cpan.org¹⁰

En nuestro primer programa de “Hola mundo”, podemos comprobar que utilizamos punto y coma (;), para separar las instrucciones.

Y, si queremos realizar comentarios en nuestro programa, utilizaremos (#).

```
#Escribimos en la pantalla  
print "Hola mundo";
```

Tipos de Operadores

Perl soporta una gran cantidad de operadores, la mayoría de ellos heredados del lenguaje C y conservan el mismo uso que en el resto de los lenguajes.

¹⁰ <http://www.cpan.org>

Operadores Aritméticos		
+	Suma	<code>\$num = \$num + \$num1;</code>
-	Resta	<code>\$num = \$num - \$num1;</code>
*	Multiplicación	<code>\$num = \$num * 5;</code>
**	Exponente	<code>\$num = 5; print "2 elevado a \$num es :", (2**\$num), "\n";</code>
/	División	<code>\$num = \$num / 8;</code>
.	Concatenación de dos o más cadenas	<code>\$cad1 = "La función chop ()"; \$cad2 = "nos sirve para quitar"; \$cad3 = "el último carácter de una cadena"; \$frase = \$cad1 . " " . \$cad2 . " " . \$cad3;</code>
%	Modulo o Residuo	<code>\$num = \$num1 % 2;</code>
x	Repetición de caracteres	<code>print "?" x 5; # imprime: ?????</code>
++	Autoincremento	<code>\$i=0; \$j=0; print \$i++; #imprime 0 y suma 1 a \$i print ++\$j; #suma 1 a \$j e imprime 1</code>
--	Autodecremento	<code>\$i=5; \$j=5; print \$i--; #imprime 5 y resta 1 a \$i print --\$j; #resta 1 a \$j e imprime 4</code>

Operadores relacionales		
	Numérico	Cadena de caracteres
Igualdad	<code>==</code>	<code>eq</code>
Diferencia	<code>!=</code>	<code>ne</code>
Inferior	<code><</code>	<code>lt</code>
Superior	<code>></code>	<code>gt</code>
Inferior o igual	<code><=</code>	<code>le</code>
Superior o igual	<code>>=</code>	<code>ge</code>

Operador cmp

Se utiliza para comparar cadenas de caracteres: Retorna **0** si las cadenas son iguales; **1** si la cadena de la derecha se encuentra al comienzo de la de la izquierda; **-1** en el caso contrario:

```
'Perl' cmp 'Perl'
# devuelve 0

'Perl me encanta ' cmp 'Perl'
# devuelve 1

'Me encanta Perl' cmp 'Perl'
# devuelve -1

'Perl' cmp 'Python'
# devuelve -1
```

Operador =~

Indica la presencia de un patrón de comparación dentro de una variable que contiene una cadena de caracteres:

```
if ($asignaturas =~/Programación/) {...}
#verifica si ' Programación' está dentro de $asignaturas
```

Operador !~

Verifica la no existencia del patrón de búsqueda en una cadena de caracteres:

```
if ($asignaturas !~/Medicina/) { ...}
#verifica que 'Medicina' no está dentro de $asignaturas
```

Operadores lógicos

Operadores lógicos		
&&	and	\$num && \$num1
	or	\$cad \$cad1
!	not	!\$cad

Estos operadores son muy usados al momento de evaluar acciones para controlar el flujo de nuestro programa, por ejemplo:

```
open(FILE,"Curriculum.txt") or die("No se pudo abrir el documento");
```

También se usan mucho para comprobar variables:

```
if(!$variable){
  print 'tienes que poner un valor';
}
```

Operadores de asignación

	Expresión	Equivalencia
=	\$num1=\$num2=\$num3;	\$num1=\$num2; \$num2=\$num3;
+=	\$num1+=3;	\$num1=\$num1 +3;
-=	\$num1-=3;	\$num1=\$num1 -3;
=	\$num1=2;	\$num1=\$num1 *2;
=	\$num1=3;	\$num1=\$num1**3;
/=	\$num1/=3+\$num2;	\$num1=\$num1/(3+\$num2)

%=	\$num1%=8;	\$num1=\$num1 %8;
.=	\$cad.="hola";	\$cad=\$cad."hola";
>>=	\$num1>>=1;	\$num1=\$num1>>1;
<<=	\$num1<<=\$num2;	\$num1=\$num1<<\$num2;
&=	\$num1&=(\$num3+=3);	\$num3=\$num3+3; \$num1=\$num1 & \$num3;
^=	\$num1^=2;	\$num1=\$num1^2;
=	\$num1 =\$num3;	\$num1=\$num1 \$num3;

Operadores a nivel de bits	
&	And bit a bit entre dos operandos
	Or bit a bit entre dos operandos
^	Or exclusivo bit a bit entre los dos operandos
~	Complemento a uno de bits. Invierte todos los bits del operando
<<	Desplazamiento a la izquierda del primer operando tantas veces como indique el segundo
>>	Desplazamiento a la derecha del primer operando tantas veces como indique el segundo

Estructuras de control.

Para poder controlar el desarrollo de un programa necesitamos instrucciones de control. Éstas permiten decidir si se ejecuta o no un bloque de instrucciones, si se hace de forma repetitiva hasta que se dé una determinada condición. Disponemos de las siguientes:

if y unless:

```

if (condición1) {
    bloque1;
}
elseif (condición2) {
    bloque2;
}
else {
    bloque3;
}

```

Si se cumple condición1, ejecuta bloque1, sino, se evalúa la condición2 y, si es cierta, se ejecuta bloque2. En caso contrario contrario, se ejecuta bloque3. Como podemos ver, si necesitamos enlazar varios if, podemos utilizar elsif.

```

unless (condición1) {
    bloque1;
}
elsif (condición2) {
    bloque2;
}
else {
    bloque3;
}

```

Si condición1 es falsa, se ejecuta bloque1. El resto son equivalentes a su significado con if.

Un ejemplo simple:

Introducimos un número: Si es menor que 0 muestra "<0"; si es menor que 5 muestra "<5"; si es mayor o igual a 5 muestra "=>5":

```

print "Introduce un número:\n";
my $numero = <STDIN>;

print "Utilizando if\n";
if ( $numero <= 0 ) {
    print "<=0\n";
}
elsif ( $numero < 5 ) {
    print "<5\n";
}
else {
    print "=>5\n";
}

print "Utilizando unless \n";
unless ( $numero > 0 ) {
    print "<=0\n";
}
else {
    unless ($numero <5) {
        print "=>5\n";
    }
    else {
        print "<5\n";
    }
}

print "Añadiendo un if al final. Si es !=0, muestra el mensaje\n";
print "Es distinto de 0\n" if ($numero!=0);

```

while y until:

```
while (condición) {
    bloque;
}
until (condición) {
    bloque;
}
```

Mediante estas instrucciones podemos ejecutar un bloque mientras se cumpla la condición (instrucción `while`) o hasta que se cumpla (instrucción `until`).

Modificamos el ejemplo anterior, permitiendo introducir números hasta que se introduzca 0:

```
my $numero = 1;
print "Empezamos con while:\n";
while ($numero != '0' ) {
    print "Introduce un número:\n";
    $numero = <STDIN>;

    if ( $numero <= 0 ) {
        print "<=0\n";
    }
    elsif ( $numero < 5 ) {
        print "<5\n";
    }
    else {
        print "=>5\n";
    }
}

$numero=1;
print "Empezamos con until\n";

until ($numero == 0 ) {
    print "Introduce un número:\n";
    $numero = <STDIN>;

    if ( $numero <= 0 ) {
        print "<=0\n";
    }
    elsif ( $numero < 5 ) {
        print "<5\n";
    }
    else {
        print "=>5\n";
    }
}
```

for:

```
for (inicio;condición;fin) {
    bloque;
}
```

```
}
```

En este bucle, *inicio* será la expresión de inicialización, *condición* que debe cumplirse para que ejecute el bloque y *fin*, la expresión que actualiza la variable de control.

Por ejemplo, un programa que escribe los números pares del 0 al 10:

```
print "Los pares del 0 al 10:\n";
for ($i=0; $i<11; $i=$i+2){
    print "$i\n";
}
```

foreach:

```
foreach $variable (@lista) {
    bloque;
}
```

`$variable` va tomando, uno a uno, todos los valores de `@lista` y ejecuta *bloque* en cada asignación. Es equivalente a `for`. Cuando queremos recorrer listas, es más elegante utilizar `foreach`.

Un ejemplo de uso de `foreach`:

```
my @semana = ("lunes","martes","miércoles","jueves","viernes","sábado","domingo");
print "\nRecorremos la lista:\n";
foreach $nombre (@semana) {
    print "$nombre ";
    foreach ($nombre) {
        /lunes/ && do { print "l \n";last;};
        /martes/ && do { print "m \n" ;last;};
        /miércoles/ && do { print "x \n";last;};
        /jueves/ && do { print "j \n";last;};
        /viernes/ && do { print "v \n";last;};
        /sábado/ && do { print "s \n";last;};
        /domingo/ && do { print "d \n";last;};
    }
}
```

El segundo `foreach` hace la función de un `if` anidado.

goto:

```
label:  
...  
goto label;
```

Esta instrucción rompe la secuencia de la ejecución y salta hasta la etiqueta *label*. Las etiquetas se definen con un nombre, seguido de dos puntos (:). No es recomendable la utilización de estos saltos.

last, next:

Estas instrucciones se utilizan para controlar la ejecución de bucles. La instrucción `last` termina la ejecución del bucle. La instrucción `next` salta el ciclo actual.

Ejemplo:

De los 10 primeros números, imprimir los primos, sin imprimir el 5:

```
for (my $i=1; $i<=10; $i++) {  
    my $primo = 1;  
    next if ($i==5);  
    for (my $j=2; $j<$i;$j++) {  
        if ( $i % $j == 0) {  
            $primo=0;  
            last;  
        }  
    }  
    print "$i es primo \n" if ($primo);  
}
```

Enlaces de interés:

<http://www.perl.com/>

<http://www.cpan.org/>



Conociendo a DOM: Parte II

Para poder desarrollar aplicaciones y sitios web con HTML5 es importante saber acceder y actualizar el contenido, la estructura y el estilo de documentos (continuación del artículo en la edición anterior)

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



Ingeniero Informático con estudios de **Master de Dirección Estratégica en TI**. Ingeniero de software en **Taller Technologies**, activista, contribuidor y consultor de proyectos **FLOSS**, miembro de **APESOL** y escritor de artículos y libros técnicos de programación.

Perfiles:

<http://about.me/neosergio>

Twitter: [@neosergio](https://twitter.com/neosergio)

Muy aparte del trabajo que se puede hacer ya con los nodos existentes, así como con sus propiedades y métodos, también pueden crearse nuevos nodos e introducirlos en cualquier parte del documento, se pueden remover, clonar y reemplazar.

Veamos que hacer en cada uno de los casos.

Recuerda que para probar código puedes usar <http://jsfiddle.net>, <http://jsbin.com> o incluso <http://dabblet.com/>.

`createElement()`

Permite la creación de un nuevo elemento con el nombre específico de etiqueta:

```
// Para crear un elemento de párrafo
parrafo = document.createElement('p');
// Para crear un elemento de división
division = document.createElement('div');
```

`createElement()` solo puede ser ejecutado con `document`, y debe ser almacenado en una variable, para

posteriormente ser insertado en el documento.

createTextNode()

Permite la creación de un nuevo nodo del tipo texto; se le pasa el texto como argumento:

```
texto = document.createTextNode("Hackers and Developers Magazine");
```

Esta función no acepta texto en formato HTML, solamente se puede usar texto plano.

appendChild()

Una vez que hemos creado elementos, debemos insertarlos en el documento, para ello podemos usar `appendChild()`, el cuál ingresa un hijo al final de la lista de hijos de un nodo. Aquí un ejemplo de como usarlo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo</title>
    <script type="text/javascript">
      function probando(){
        parrafo = document.createElement('p');
        texto = document.createTextNode('Hackers and Developers');
        parrafo.appendChild(texto);
        document.getElementById('saludo').appendChild(parrafo);
      };
    </script>
  </head>
  <body>
    <section id="saludo">
      <p>Este es un ejemplo</p>
    </section>
    <h1 onClick='probando()'>Clic aquí para probar. Sí, dale clic aunque
    no parezca un enlace</h1>
  </body>
</html>
```

insertBefore()

Es similar a `appendChild()` excepto por el orden de inserción ya que con esta función se le puede especificar el elemento que se posicionará luego del nuevo elemento, es decir antes del nodo especificado:

```
<!DOCTYPE html>
<html>
```

```

<head>
  <title>Ejemplo</title>
  <script type="text/javascript">
    function probando(){
      parrafo = document.createElement('p');
      texto = document.createTextNode('Hackers and Developers');
      parrafo.appendChild(texto);
      anterior = document.getElementById('primer');
      document.getElementById('saludo').insertBefore(parrafo, anterior);
    };
  </script>
</head>
<body>
  <section id="saludo">
    <p id="primer">Este es un ejemplo</p>
  </section>

  <h1 onClick='probando()>Clic aquí para probar. Sí, dale clic aunque
  no parezca un enlace</h1>
</body>
</html>

```

cloneNode()

Permite copiar un nodo de dos maneras que podríamos llamar parcial y completa (incluyendo a sus respectivos hijos):

```

//copia parcial, quiere decir que solo copiara la división sin hijos
division = document.getElementById('division1').cloneNode(false);
//copia completa, copiara la división incluyendo todos sus hijos
division = document.getElementById('division1').cloneNode(true);

```

De manera predeterminada la función hace copias parciales y una vez copiados los elementos ya se pueden insertar en el lugar donde queramos.

removeChild()

Permite remover nodos hijos:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo</title>
    <script type="text/javascript">
      function probando(){
        primer = document.getElementById('primer');
        document.getElementById('saludo').removeChild(primer);
      };
    </script>
  </head>
  <body>
    <section id="saludo">
      <p id="primer">Este es un ejemplo</p>
      <p id="segundo">Este es un segundo parrafo</p>
    </section>
  </body>
</html>

```

```

</section>

<h1 onClick='probando()>Clic aquí para probar. Sí, dale clic aunque
no parezca un enlace</h1>
</body>
</html>

```

Al ejecutar el ejemplo, el párrafo “Este es un ejemplo” debe desaparecer.

replaceChild()

Reemplaza un nodo antiguo por uno nuevo:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo</title>
    <script type="text/javascript">
      function probando(){
        texto = document.createTextNode('Hackers and Developers');
        reemplazo = document.createElement('p').appendChild(texto);
        primer = document.getElementById('primer');
        seccion = document.getElementById('saludo');
        seccion.replaceChild(reemplazo, primer);
      };
    </script>
  </head>
  <body>
    <section id="saludo">
      <p id="primer">Este es un ejemplo</p>
      <p id="segundo">Este es un segundo parrafo</p>
    </section>

    <h1 onClick='probando()>Clic aquí para probar. Sí, dale clic aunque
no parezca un enlace</h1>
  </body>
</html>

```

Y por último aquí un ejemplo combinando algunos métodos aprendidos anteriormente:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Creando contenido</title>
    <script type="text/javascript">
      function creando(){
        //creando el contenido
        titulo = document.createTextNode('Titulo');
        texto = document.createTextNode('Hackers and Developers');
        autor = document.createTextNode('por: Sergio Infante');
        //creando los elementos y poniéndoles el contenido
        titulo_html = document.createElement('h1');
        titulo_html.appendChild(titulo);
        texto_html = document.createElement('p');

```

```

    texto_html.appendChild(texto);
    autor_html = document.createElement('h2');
    autor_html.appendChild(autor);
    //insertando los elementos
    saludo = document.getElementById('saludo');
    saludo.appendChild(titulo_html);
    saludo.appendChild(autor_html);
    saludo.appendChild(texto_html);
};
</script>
</head>
<body>
  <section id="saludo">

  </section>

  <h1 onClick='creando()>Clic aquí para probar. Sí, dale clic aunque
  no parezca un enlace</h1>
</body>
</html>

```

Como habrán podido darse cuenta, al leer en estas dos partes del artículo «*Conociendo a DOM*», es de mucha ayuda entender como funciona el DOM para manipular su comportamiento y así ser exitosos al hacer desarrollo web.

Una completa descripción del DOM puede ser revisada en el Mozilla Developer Network¹¹.

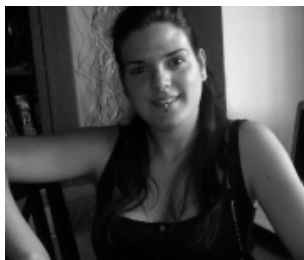


11 <https://developer.mozilla.org/en-US/docs/DOM>

Pásate a GNU/Linux con Arch: Pacman, el gestor de paquetes

Algo que debemos conocer de una distribución GNU/Linux es cómo se gestiona la paquetería. ¿Cómo podemos instalar, desinstalar o ver información sobre los paquetes instalados?

Escrito por: **María José Montes Díaz** (Archera & 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://archninfo.blogspot.com/es/>

Redes sociales:

Twitter: [@MMontesDiaz](https://twitter.com/MMontesDiaz)

Este gestor de paquetes está escrito en C. Utiliza archivos empaquetados en tar y comprimidos en gzip o xz para todos los paquetes. Cada uno contiene los binarios compilados. Los paquetes son descargados a través de FTP, HTTP o archivos locales, dependiendo de cómo esté configurado cada repositorio. Puede actualizar el sistema mediante un solo comando.

No sólo es utilizado por Arch (aunque nació para ser el gestor de esta distribución), también se utiliza en Chakra, Destro, Manjaro y Deli Linux. FrugalWare utiliza un *fork* de pacman.

La configuración de pacman se encuentra en el archivo `/etc/pacman.conf`. En este archivo podemos establecer las opciones especiales y especificar los repositorios para pacman.

Principales comandos de pacman

Podemos dividirlos en tres bloques: Instalación, Desinstalación y Consultas:

Instalación:

```
$sudo pacman -S lista_de_paquetes    .- Instala la lista de paquetes.
$sudo pacman -R lista_de_paquetes    .- Elimina la lista de paquetes.
$sudo pacman -Rs lista_de_paquetes   .- Elimina la lista de paquetes y sus
                                     dependencias.
$sudo pacman -Syu                     .- Sincroniza y actualiza todos los paquetes.
$sudo pacman -Syu lista_de_paquetes  .- Actualiza el sistema e instala la lista de
                                     paquetes.
$sudo pacman -Sw lista_de_paquetes   .- Descarga la lista de paquetes sin
                                     instalarlos.
$sudo pacman -U /ruta/nombrepaquete  .- Instala un paquete local.
$sudo pacman -U http://w.ejemplo.com/rep/paquete1 .- Instala un paquete remoto.
$sudo pacman -Sf                      .- Fuerza la instalación del paquete.
```

Desinstalación:

```
$sudo pacman -R lista_de_paquetes    .- Desinstala la lista de paquetes.
$sudo pacman -Rs lista_de_paquetes   .- Desinstala la lista de paquetes y sus
                                     dependencias.
$sudo pacman -Rsc lista_de_paquetes  .- Desinstala la lista de paquetes, sus
                                     dependencias y aquellos paquetes que
                                     dependen de alguno de la lista generada.
$sudo pacman -Rn lista_de_paquetes   .- Pacman salva algunos archivos de los paquetes
                                     al ser desinstalados si han sido modificados
                                     por el usuario. Con esta opción evitamos que
                                     se generen.
```

Instalación/desinstalación:

```
$sudo pacman -[S|U|R]d               .- Se saltan los controles de dependencia de
                                     versión. Los nombres de paquetes se siguen
                                     comprobando.
$sudo pacman -[S|U|R]dd              .- Para saltar todas las comprobaciones de dependencia.
```

Las actualizaciones parciales no están soportadas. No utilizar pacman -Sy lista_paquetes, ni instalar paquetes si se ha hecho un pacman -Sy y no se ha actualizado el sistema.

Consultas:

```
$pacman -Ss cadena_1 cadena2...  .- Buscar paquetes en la base de datos.
$pacman -Q                        .- Muestra una lista de todos los paquetes
                                instalados en el sistema.
$pacman -Qs cadena_1 cadena2...  .- Buscar paquetes dentro de los paquetes
                                instalados.
$pacman -[S|Q]i paquete          .- Muestra información sobre determinado paquete.
$pacman -Qii paquete             .- Muestra información sobre un paquete instalado y
                                lista de archivos de copia de seguridad y sus
                                estados.
$pacman -Ql paquete              .- Para obtener la lista de archivos instalados por
                                un paquete.
$pacman -Qo /ruta/archivo        .- Para obtener el nombre del paquete que instalo
                                el archivo.
$pacman -Qdt                     .- Para obtener una lista de todos los paquetes
                                instalados como dependencias no requeridas.
```

Limpieza de la caché

Según vamos actualizando/instalando paquetes, éstos se van almacenando en la caché de pacman. Ésta va creciendo, dejando la partición dónde se encuentre /var montado sin espacio, dado que los paquetes descargados se encuentran /var/cache/pacman/pkg.

Pacman nos ofrece un comando para eliminar los paquetes antiguos guardados en la caché:

```
$sudo pacman -Sc  .-Borra todos los paquetes antiguos guardados en la cache pacman.
                  Mantiene la versión actual instalada y elimina todos los que no
                  estén instalados.
$sudo pacman -Scc .-Borra todos los paquetes guardados en la cache pacman.
```

Para un tratamiento de la caché más selectivo, podemos utilizar paccache, que forma parte del paquete pacman-contrib. Primero instalamos pacman-contrib:

```
$sudo pacman -S pacman-contrib
```

Ahora podemos utilizar el comando:


```
$sudo paccache -rv
```

que eliminará todos los paquetes de la caché, pero mantendrá las tres últimas versiones (este es el valor por defecto, con el parámetro -k podemos modificarlo).

No elimines toda la caché, mantén al menos dos versiones para poder realizar un **downgrade**, en caso de que sea necesario.

Scripts y consejos para el mantenimiento

Después de actualizar pacman, es conveniente actualizar la base de datos y optimizarla. Para ello podemos ejecutar:

```
$sudo pacman-db-upgrade && sudo pacman-optimize
```

Una de las cosas que suele pasar al desinstalar aplicaciones o al actualizar el sistema, es que varíen las dependencias o queden dependencias en nuestro sistema que ya no están siendo utilizadas por ningún paquete. A estos paquetes se les llama "huérfanos": Fueron instalados porque otro paquete los requería pero ya no son necesarios.

Para eliminar los paquetes huérfanos, podemos utilizar el siguiente comando:

```
$sudo pacman -Rs $(pacman -Qtdq)
```

Podemos crear un alias en nuestro ~/.bashrc o añadir una función en el mismo para la eliminación de los paquetes huérfanos:

Opción 1:

```
alias huerfanos="sudo pacman -Rs $(pacman -Qtdq)"
```

Opción 2:

```
huerfanos() {  
  if [ ! -n $(pacman -Qdt) ]; then  
    echo "No hay nada que hacer."  
  else  
    sudo pacman -Rs $(pacman -Qdtq)  
  fi  
}
```

```
}
```

Ahora sólo debemos ejecutar:

```
$ huerfanos
```

Cómo hemos utilizado `sudo` tanto para el alias cómo para la función, no es necesario ejecutarlo como `root` ya que nos pedirá la clave cuándo sea necesario.

Las dependencias opcionales, si las instalamos con la opción `--asdep`, `pacman` las tratará como paquetes huérfanos. Si vamos a instalar dependencias opcionales, no debemos utilizar esa opción. Conviene revisar la lista de paquetes generada antes de confirmar su eliminación.

Para identificar los archivos que tenemos en el sistema que no pertenecen a ningún paquete, podemos crear el *script* `/usr/local/bin/pacman-disowned`, con el siguiente contenido:

```
#!/bin/sh
tmp=${TMPDIR-/tmp}/pacman-disowned-$UID-$$
db=$tmp/db
fs=$tmp/fs
mkdir "$tmp"
trap 'rm -rf "$tmp"' EXIT
pacman -Qlq | sort -u > "$db"
find /bin /etc /lib /sbin /usr/bin /usr/share\
  ! -name lost+found \
  \( -type d -printf '%p/\n' -o -print \) | sort > "$fs"
comm -23 "$fs" "$db"
```

Lo hacemos ejecutable:

```
$ sudo chmod +x /usr/local/bin/pacman-disowned
```

Podemos generar una lista con estos archivos así:

```
$ sudo pacman-disowned > lista.txt
```

Antes de proceder a su eliminación, deberemos verificar a qué corresponden, pues pueden ser archivos de configuración personalizados, *logs*, etc.

En el caso de que queramos dejar la instalación limpia, sólo con el grupo base, podemos

utilizar el siguiente comando:

```
$sudo pacman -Rs $(comm -23 <(pacman -Qeq|sort) <((for i in $(pacman -Qqg base); do  
pactree -ul $i; done)|sort -u|cut -d ' ' -f 1))
```

Algo que puede sernos muy útil para hacer copias de seguridad, es conocer qué archivos de configuración hemos modificado. Así no es necesario hacer un *backup* de toda la carpeta */etc*, tan sólo de aquellos que nos interese.

Para saber qué archivos de configuración hemos modificado, podemos crear un *script* llamado */usr/local/bin/pacman-changed-files*, con el siguiente contenido:

```
#!/bin/bash  
for package in /var/lib/pacman/local/*; do  
    sed '/^%BACKUP%$/,/^%/!d' $package/files | tail -n+2 | grep -v '^$' | while  
    read file hash; do  
        [ "$(md5sum /$file | (read hash file; echo $hash))" != "$hash" ] &&  
    echo $(basename $package) /$file  
    done  
done
```

Lo hacemos ejecutable:

```
$sudo chmod +x /usr/local/bin/pacman-changed-files
```

Debemos ejecutarlo con permisos de administrador.

Cada vez que actualicemos el sistema, si se han creado o modificado los servicios de *systemd*, debemos recargar la lista de unidades de servicio. Esto podemos automatizarlo de la siguiente manera:

Creamos la unidad de servicio */usr/local/lib/systemd/system/reload-daemon.service* y añadimos:

```
[Unit]  
Description=Recarga la lista de unidades de servicio cuándo se modifican.  
  
[Service]  
Type=oneshot  
ExecStart=/usr/bin/systemctl --system daemon-reload  
RemainAfterExit=no
```

Creamos */usr/local/lib/systemd/system/daemon-reload.path*

```
[Unit]
```

```
Description=Recarga la lista de unidades de servicio cuándo se modifican.
```

```
[Path]
PathChanged=/usr/lib/systemd/system
PathChanged=/usr/local/lib/systemd/system
Unit=daemon-reload.service
```

```
[Install]
WantedBy=multi-user.target
```

Para iniciarla:

```
$sudo systemctl start daemon-reload.path
$sudo systemctl enable daemon-reload.path
```

Una vez vistos los comandos para pacman, debemos tener en cuenta una serie de cosas:

- 1.- Archlinux es *Rolling Release*.** Esto quiere decir que hay actualizaciones a diario. En cuanto un paquete es lo suficientemente estable, pasa a los repositorios. En el caso de bibliotecas, todos los paquetes asociados son recompilados. Esta característica hace que no podamos realizar actualizaciones parciales (`#pacman -Sy <lista de paquetes>`). Si es necesario sincronizar la base de datos antes de instalar, se debe actualizar el sistema. Para no equivocarnos, siempre que necesitemos realizar una sincronización de paquetes, es mejor hacer un `pacman -Syu`.
- 2.-** Dado que es una distribución que se actualiza muy rápidamente, a veces pueden surgir inconvenientes. No es recomendable realizar actualizaciones del sistema si se necesitan tener el sistema estable. Es decir, es mejor actualizar cuándo se tenga tiempo libre.
- 3.-** Es conveniente no eliminar la caché de pacman nada más actualizar. Si necesitamos hacer un downgrade, no podremos hacerlo a partir de la caché. Se podría hacer a partir de las imágenes que hay en la [Arch Rollback Machine](#)¹². Aún así, no es buena práctica eliminar completamente la caché.
- 4.-** Evitar la utilización del comando `--force` (`pacman -Sf <lista de paquetes>`). No utilizarlo nunca en actualizaciones completas del sistema (`pacman -Syuf`). Si en una actualización completa pacman da error, es conveniente revisar las [News](#)¹³. En el caso de necesitar algún tipo de intervención por parte del usuario, se publicará allí la solución.
- 5.-** Cómo consejo general, no actualizar el sistema sin antes haber revisado las [News](#).

¹² <http://arm.konnichi.com/>

¹³ <https://www.archlinux.org/news/>

Backup automático de la base de datos de Pacman

La base de datos de pacman se encuentra en `/var/lib/pacman/local`. Es una buena idea tener una copia de seguridad de la misma. Si por algún motivo se corrompe, podremos restaurarla.

Un método que permite hacerlo automáticamente, es crear un servicio para `systemd` que se encargue de realizar la copia cada vez que se modifique la base de datos.

Los *scripts* de `systemd` se suelen crear en `/usr/lib/systemd/scripts` (para los paquetes instalados) o en `/usr/local/lib/systemd/scripts` (para los creados por el usuario). Como estos archivos los vamos a crear nosotros, vamos a crear la ruta para almacenar los archivos:

```
$sudo mkdir -p /usr/local/lib/systemd/scripts
$sudo mkdir -p /usr/local/lib/systemd/system
```

Ahora creamos el *script* `/usr/local/lib/systemd/scripts/pakbak_script` y le añadimos el siguiente código:

```
#!/bin/bash
declare -r pakbak="/pakbak.bz2"
tar -cjf "$pakbak" "/var/lib/pacman/local"
```

En la variable `pakbak` especificamos dónde queremos guardar la copia de seguridad. Debemos darle permisos de ejecución:

```
$sudo chmod +x /usr/local/lib/systemd/scripts/pakbak_script
```

Creamos el servicio: `/usr/local/lib/systemd/system/pakbak.service` y le añadimos:

```
[Unit]
Description=Back up pacman database

[Service]
Type=oneshot
ExecStart=/bin/bash /usr/local/lib/systemd/scripts/pakbak_script
RemainAfterExit=no
```

Y por último, creamos el servicio `/usr/local/lib/systemd/system/pakbak.path`, que estará vigilando la carpeta `/var/lib/pacman/local`, que es dónde está la base de datos:

```
[Unit]
```

```
Description=Back up pacman database
```

```
[Path]  
PathChanged=/var/lib/pacman/local  
Unit=pakbak.service
```

```
[Install]  
WantedBy=multi-user.target
```

Ahora, para iniciar el servicio:

```
$sudo systemctl start pakbak.path
```

Para habilitar el inicio en cada arranque:

```
$sudo systemctl enable pakbak.path
```

Configurando pacman

El comportamiento de pacman y los repositorios se encuentra en el archivo `/etc/pacman.conf`. Este archivo está dividido en dos partes: opciones generales y repositorios.

Veamos algunas directivas interesantes:

```
HoldPkg = paquetel paquete2 ... .- Si se intenta desinstalar algún paquete perteneciente a la lista indicada, pide confirmación.
```

```
SyncFirst = paquetel paquete2 ... .- Lista de paquetes que se deben actualizar primero, antes de actualizar el sistema completo. Por defecto se encuentra pacman, es posible que queramos añadir yaourt y package-query.
```

```
IgnorePkg = paquetel paquete2 ... .- No actualiza la lista de paquetes indicada.
```

```
IgnoreGroup = grupo1 grupo2 ... .- No actualiza la lista de grupos indicada.
```

```
NoUpgrade = archivol archivo2 ... .- Al actualizar el paquete, no reescribe ni crea los .pacnew correspondientes a esos archivos. Hay que tener en cuenta que no se ha de especificar la ruta anteponiendo /, por ejemplo, para no actualizar /etc/fstab, deberemos poner etc/fstab.
```

```
NoExtract = archivol archivo2 ... .- Al actualizar, estos archivos no son sobrescritos. Se crea en la ruta de cada uno un archivo con el mismo nombre que el original y con la extensión .pacnew.
```

```
XferCommand = /ruta/commando %u .- Permite cambiar el gestor de descargas de pacman por otro.
```

Ahora veamos el formato para los repositorios:

```
[repo-name]
SigLevel = Package{Required|Opcional|Never}
Server = ServerName
Include = IncludePath
```

Primero, debemos especificar el nombre, en este caso, repo-name. En Server especificamos el servidor. Si es un repositorio que está en varios servidores, podemos especificar el archivo donde están estos servidores mediante la cláusula Include.

Con la cláusula SigLevel establecemos cómo se gestionará la firma de paquetes:

PackageRequired: Los paquetes han de estar firmados.

PackageOpcional: Si los paquetes están firmados, se comprueba la firma, si ésta falla, no se instala. Si no existe firma, no se considera error y se instala.

PackageNever: No se comprueba la firma de paquetes.

Debemos tener en cuenta que el orden de los repositorios importa. El paquete a instalar se buscará en los mismos por orden de aparición en pacman. Desde el primer repositorio que lo contenga, será desde el que actualice o instale el paquete. Si en un repositorio posterior, el paquete se encuentra con una versión superior, no se actualizará.

Enlaces de interés:

<https://wiki.archlinux.org/index.php/Pacman>

<https://www.archlinux.org/pacman/pacman.8.html>

MARÍA JOSÉ MONTES DÍAZ TE INVITA A PARTICIPAR DEL PROYECTO E-CIDADANIA. Pulsa sobre la imagen para informarte o visita los enlaces que figuran a continuación.



BETA

e-cidadania
democracia participativa ciudadana

 open source

 GPL v3

<http://ecidadania.org> | <http://code.ecidadania.org> | <http://docs.ecidadania.org>

La experiencia con ssh sobre redes móviles puede ser dolorosa. Con Mosh podemos olvidarnos de ver con cierta frustración el mensaje «*Write failed: Broken pipe*» cada vez que nuestro portátil va "a dormir", cuando se presenta una intermitencia en la red o simplemente, cambiamos de red.

Escrito por: **Elizabeth Ramírez** (Ingeniera Electrónica)



Desarrolladora de Software en las Industrias Web, Telecomunicaciones y Entretenimiento. Especialista en Sistemas de Información, con énfasis en Real-Time Billing. Miembro de IEEE Signal Processing Society, New York Section. "Wantrepreneur".

Webs:

About.Me: <http://about.me/elizabethr>

Redes sociales:

Twitter: [@eramirem](https://twitter.com/eramirem)

Mosh (abreviatura de *mobile shell*) deriva su nombre de quizás su característica más notable y es el soporte de *roaming* y conexiones intermitentes, problemas intrínsecos a redes inalámbricas y móviles. Esto significa que Mosh mantendrá "viva" una sesión segura, establecida con un servidor remoto aún, si se pierde la conexión a la red. Lo anterior se traduce en que la sesión es restablecida cuando la conexión es recuperada; sin embargo, mientras la conexión a la red se encuentre caída, no se producen ecos locales, como podría esperarse.

Entre otras características secundarias, Mosh es muy cuidadoso con los comportamientos extraños que se presentan en los emuladores de terminal cuando se ingresan secuencias de caracteres Unicode o ISO 2022, que pueden bloquearla.

"ISO 2022 locking escape sequences oh flying spaguettin monster please kill me now." Revisor de USENIX leyendo el paper de Mosh.

Mosh está escrito en C++ y Software Libre, distribuido bajo licencia GPL v 3.

¿Cómo funciona Mosh?

En los laboratorios de Computer Science e Inteligencia Artificial del MIT, han ideado un nuevo protocolo llamado SSP (*State Synchronization Protocol*), el cual opera sobre UDP, en lugar de operar sobre TCP como lo hace SSH. Para aquellos que lo desconozcan, TCP es un protocolo orientado a la conexión, es decir, la conexión debe ser establecida entre ambas partes antes de empezar a transmitir paquetes, de manera opuesta a UDP, que es no-orientado a la conexión y el flujo de datagramas puede transcurrir sin que se haya establecido la conexión.

Es precisamente esta característica la que permite a Mosh minimizar la sobrecarga de datos sobre el canal y funcionar en modo predictivo, teniendo a su vez mecanismos para reordenar y manejar paquetes repetidos.

En el funcionamiento fundamental de ssh, al establecer una sesión, el *echo* o efecto de cada evento del teclado en la terminal, es generado por el servidor. Cada entrada del

teclado viaja hasta el servidor de ssh y regresa al terminal. En el caso de Mosh, el cliente genera un *echo* local de manera predictiva para cada entrada. Este modo predictivo permite que el mosh-client prediga el efecto en el terminal de cada entrada del teclado y lo aplique inmediatamente, en lugar de ir hasta el servidor y de tal manera que cuando la comunicación entre el cliente y el servidor tiene una alta latencia o no es transmitida de manera uniforme, es transparente para el usuario, conllevando a la disminución de errores de teclado.

Mosh está habilitado para recuperar la sesión automáticamente, pues permite almacenar el estado más reciente de la terminal en el nodo local y remoto de la conexión. Mosh confía en la historia de entradas y el contenido de la ventana del terminal cuando la red es intermitente, para realizar especulaciones sobre el efecto de cada entrada en la terminal en dichos casos.

La prueba

Una de las desventajas (a título personal) de mosh, es que debe ser instalado tanto en el cliente como en el servidor.

Para validar el funcionamiento de mosh, se usaron dos instancias de un servicio Web que provee herramientas de flexibles¹⁴ de *cloud computing* con Ubuntu Server 12.04. Para realizar dicha prueba, debió agregarse una regla Custom UDP en Security Groups, que permita conexiones entrantes al rango de puertos 60000 – 61000, en la instancia del servidor.

Los pasos de instalación ejecutados fueron los siguientes¹⁵:

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository ppa:keithw/mosh
$ sudo apt-get update
$ sudo apt-get install mosh
```

Del lado del servidor, Mosh se autentica mediante ssh, abre un *listener* en un puerto

14 NdR: el nombre comercial de dicho servicio ha sido eliminado por ser privativo (opera con Software no-libre) y contener restricciones legales que dificultan su publicación en un medio de comunicación.

15 Los detalles de instalación para múltiples sistemas operativos se encuentran en <http://mosh.mit.edu/#getting>

UDP y genera una llave aleatoria AES-128 compartida, la cual será usada por el cliente para autenticarse.

```
ubuntu@ip-10-xxx-xx-xxx:~$ mosh-server

MOSH CONNECT 60001 LX712VdKKUH+AF8FZIqIfw
ubuntu@ip-10-xxx-xx-xxx:~$
mosh-server (mosh 1.2.3)
Copyright 2012 Keith Winstein <mosh-devel@mit.edu>
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

[mosh-server detached, pid = 4250]

ubuntu@ip-10-xxx-xx-xxx:~$
```

Del lado del cliente, exportamos la llave generada y especificamos la dirección IP del servidor y puerto del listener de mosh.

```
$ export MOSH_KEY=LX712VdKKUH+AF8FZIqIfw
$ mosh-client 12.345.67.890 60001

Welcome to Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-31-virtual i686)

ubuntu@ip-12-345-67-890:~$
```

Se realizaron básicamente 3 pruebas:

- La primera consistió en desconectar la red Wi-Fi y volverla a conectar transcurrido un tiempo.
- La segunda consistió en enviar “a dormir” el equipo en el que se encuentra el cliente con la sesión de mosh establecida.
- La tercera prueba consistió en cambiar de una red Wi-Fi a una red 4G. Mientras una sesión ssh se cierra en las tres pruebas, voilá! la sesión mosh permanece viva.

Conclusión

La sensación que da Mosh al usuario, es que se está trabajando en un equipo local. Una prueba interesante sería utilizar Mosh sobre una red muy degradada y observar los tiempos de respuesta de los *echoes*, significativamente mas cortos que al usar ssh, al estar operando sobre UDP. Para conocer la información del proyecto en detalle, visitar <http://mosh.mit.edu>

Se espera que en un futuro, mosh soporte IPv6.

Agilismo en palabras simples

Se menciona mucho el termino ágil y metodologías ágiles en los últimos tiempos; es importante entender la esencia del mismo.

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



Ingeniero Informático con estudios de **Master de Dirección Estratégica en TI**. Ingeniero de software en **Taller Technologies**, activista, contribuidor y consultor de proyectos **FLOSS**, miembro de **APESOL** y escritor de artículos y libros técnicos de programación.

Perfiles:

<http://about.me/neosergio>

Twitter: [@neosergio](https://twitter.com/neosergio)

Nada pone mas feliz a un cliente, que el software que ha encargado, esté funcionando; que todo el dinero y recursos invertidos, den fruto y que pueda evidenciar directamente el producto del trabajo. Esto es básicamente lo que busca el agilismo, hacer al cliente feliz, en el menor tiempo posible.

Pongámonos en el zapato del cliente; olvidemos que estamos dedicados a construir software; imaginemos que es nuestro dinero y contratamos a un equipo para llevar a cabo el proyecto: ¿qué nos daría confianza en el equipo? ¿mucha documentación, planes y reportes? o ¿Software que funciona y que va creciendo en funcionalidades, semana a semana?

Cuando se empieza a ver el desarrollo de software desde la perspectiva del cliente, es cuando realmente empiezan a pasar cosas buenas.

Para poder entender mejor al agilismo veamos la parte más importante del manifiesto ágil¹⁶ firmado hace 12 años, el 17 de febrero del 2001, por 17 desarrolladores de software que se reunieron para participar de una convención sobre los métodos ligeros de desarrollo.

Individuos e interacciones sobre procesos y herramientas |

¹⁶ <http://agilemanifesto.org/iso/es/>

Estos lineamientos se complementan con los doce principios de las metodologías ágiles.¹⁷

Lo bueno

De los proyectos en los cuales he podido participar y aplicar las metodologías ágiles, he podido resaltar las siguientes razones por las cuáles ser ágil:

1. **Enfocarse en lo que es realmente importante y postergar lo que no:** el agilismo te hace dejar de lado aquellas actividades que no van a contribuir a entregar software funcionando en el menor tiempo, te hace pensar en qué es lo que va a hacer al cliente feliz.
2. **Dividir grandes problemas inmanejables en problemas pequeños fáciles de solucionar,** incluyendo una mejora en el tiempo para solucionar problemas. En relación a la razón anterior, si uno se enfoca en lo que realmente importa, un problema pequeño puede ser dividido en partes de acuerdo a su importancia.
3. **Estar seguro de que realmente funciona lo que estas entregando al cliente:** al ser entregas en corto tiempo y pensando en que debe ser lo mas importante para el cliente, incluso si llega a fallar lo que entregas o no se adapta a lo que el cliente desea, es rápido corregirlo, puesto que el cliente te da la retroalimentación para hacerlo. Los problemas aparecen más rápido y obviamente las soluciones.
4. **Disfrutar del poder de la retroalimentación,** al ser periodos cortos enfocados a entregar versiones funcionales del software, los comentarios, opiniones y defectos que nota el equipo, cliente y usuarios llegan muy rápido. Esto permite ajustar a las necesidades del caso.
5. **Volverse medible:** el cliente se da cuenta como está siendo utilizado su dinero, de la calidad del trabajo, del cronograma planificado y de manejar las expectativas.
6. **Volverse flexible al cambio:** en un proyecto pueden pasar muchas cosas que nos obligan a hacer cambios, es importante tener la ventaja de cambiar los planes cuando sean necesarios y adaptarse al contexto del proyecto.

La advertencia

A pesar de todas las razones buenas, las cuales muchos de los que practican las metodologías ágiles pueden corroborar, no a todos les gusta esta forma de trabajo, por la misma razón que muchas personas no comen saludablemente o hacen ejercicios.

¹⁷ <http://www.agilemanifesto.org/principles.html>

El entregar valor cada semana, es una actividad que requiere de transparencia en el proceso de construcción; no hay lugar para ocultar cosas: o se entrega algo de valor o no se entrega, es tan simple como eso.

Sin embargo si te gusta la transparencia, te apasionas por hacer un trabajo de alta calidad y tomas la decisión de ejecutar y trabajar, las metodologías ágiles funcionarán como una recompensa y hasta una manera divertida de construir software.

No hay obligación por hacer entregas cada semana si piensas que puede ser estresante. Muchos equipos ágiles empiezan a entregar cada dos semanas o incluso tres semanas, no lo tomes literalmente, es solo un decir para hacerte entender lo beneficioso que puede ser entregar versiones funcionales de software al cliente, obtener retroalimentación y cambiar los planes si son necesarios.

La prioridad mas alta es satisfacer al cliente a través de la entrega pronta y continua de software útil.

La confusión

Debido al éxito que tienen las metodologías ágiles, hay ciertas acciones que han hecho que los escépticos le tengan mucha resistencia y con mucha razón se pueden encontrar hechos como los siguientes:

- **Las empresas siguen viviendo el proceso en cascada:** adaptarse al agilismo es un cambio que incluye también la manera de pensar y la cultura *organizacional*.
- **Falsas expectativas causadas por creer que es la bala de plata** la que soluciona todos los problemas, la que nunca fallará.
- **Considerar a las metodologías ágiles como un proceso** y poner ese proceso y la elección de las mejores herramientas sobre las necesidades del equipo (va en contra de la primera parte del manifiesto ágil).
- **Las certificaciones que pretenden hacer creer que obteniéndose ya se convierten automáticamente en *agilistas* expertos:** hay un negocio muy grande con el término *scrum master* y cursos sobre eso.
- **Las malas capacitaciones y entrenamientos** dadas por académicos que intentan conocer los detalles de las metodologías ágiles pero que jamás han hecho software.
- **Las implementaciones estrictas de las metodologías,** porque las consideran cómo algo muy ceremonial, dejando de lado lo pragmático.
- **Usar post-it en un pizarra** y pensar que con implementar eso ya son ágiles y están preparados para mejorar el proceso.
- **La mala asociación del termino ágil** con Scrum, Kanban, eXtreme Programming que son solo formas de trabajo, creando incluso fanatismo por ellos, pensando que seguir sus procesos al pie de la letra ya son *agilistas*.

- **Crear que ágil significa no documentar.**
- **Haber convertido al agilismo en una especie de religión,** promoviendo el fanatismo y buscando atacar y discrepar con todo lo demás.
- **Confundir a los clientes con terminologías ágiles** y pretender que entiendan estos términos y forzarlos a ser supuestamente ágiles.

Debido a estas razones y algunas otras más, se está hablando de la era del **post-agilismo**. Algunos de los artículos y presentaciones más resaltantes al respecto son las publicadas por *Javier Garzás: La agilidad esta muriendo. Bienvenido el postagilismo*¹⁸, la de *Israel Alcázar* con el título *Desmitificando el agilismo*¹⁹ y la de *Miguel Ángel Garcia: Agile ha muerto. ¡Viva post-agile!*²⁰

La planificación

Planificar un proyecto ágil no difiere mucho de tener una lista de cosas por hacer y tareas, a todo esto se le llama Lista de historias e historias de usuario.

La lista de historias es la lista de cosas por hacer en el proyecto. Contiene todas las características que el cliente quiere ver en su software; está ordenada de acuerdo a la prioridad de necesidades del cliente y validada por el equipo de desarrollo; es la base del plan de proyecto.

La iteración es un periodo de una a dos semanas donde se toman las características más importantes para el cliente y se transforman en realidad. Eso permite determinar la velocidad de desarrollo (cantidad de características por cada iteración). Así se va conociendo mejor el equipo y puede ir mejorando su desempeño.

Cuando no queda tiempo para terminar las tareas en cada iteración, se hacen menos tareas, siempre en orden de importancia y cuando las cosas van realmente mal, se cambia el plan, pues la planificación es flexible.

No hay planificación mágica desde el principio que permita que sucedan milagros en el camino y se tenga el software útil y listo para funcionar. En el agilismo no se necesitan milagros, ya que todo es honesto con los clientes desde el principio.

Imaginemos lo siguiente: Debemos contratar a una persona para que limpie las ventanas de nuestra casa, ¿crees que deberíamos considerar el trabajo hecho? cuando esta persona nos presente:

- Nos muestra un reporte de como planea limpiar las ventanas.
- Viene con un elaborado mapa de ruta de como limpiara.
- Crea un elaborado plan de pruebas para saber que las ventanas están limpias.

De ninguna manera esta persona recibiría un pago, hasta que las ventanas estén limpias.

En las metodologías ágiles, sucede algo similar, entregar una característica al cliente

18 <http://www.javiargarzas.com/2012/05/postagilismo.html>

19 <http://farmerdev.com/slides/2012/desmitificando-agile/index.html>

20 <http://magmax.org/2012/04/16/post-agile.html>

significa hacer todo lo necesario para producir esa característica y que sea útil (analizar, probar, diseñar, construir, documentar y todo lo demás). Esto no significa que necesariamente todo esté en la primera versión, pero sí que se ponga todo el esfuerzo como si se tuviera que hacer.

El software funcionando y útil es la principal forma de medir el éxito

La verdad

Cuando se trabaja con metodologías ágiles hay verdades con las que se van a topar y es importante respetarlas:

- **Es imposible captar todos los requerimientos del cliente al principio del proyecto:** no hay que tener miedo de empezar el proyecto sin saber todo lo que viene mas adelante. Los requerimientos seguirán escondidos hasta que se inicie el proyecto.
- **Los requerimientos que identificaste siempre van a cambiar,** no hay que tener miedo al cambio, acéptalo y adapta el plan cuando sea necesario.
- **Siempre habrá mas cosas que hacer que el dinero y tiempo disponible:** no te estreses cuando la lista de cosas por hacer excede los recursos, es normal, sólo haz las cosas que realmente puedas, ordena por prioridad la lista y deja las cosas menos importantes para el final.
- **No existe una sola forma de hacer las cosas:** no te estreses por elegir la mejor metodología. Imagina a Scrum, XP, Kanban, Lean, entre otras, como si fueran sabores de helados, siempre habrá alguien que dirá que ese sabor es mejor que otro, pero no es cierto, depende de los gustos, preferencias y necesidades. No existe metodología que te de todo lo que necesitas. Cada proyecto es diferente, puedes combinarlas, aplicar principios y prácticas e incluso crear tu propia metodología.

Las metodologías ágiles están orientadas a hacer a las personas (clientes y desarrolladores) felices.



¿nos echarías una mano?

PayPal

VISA MasterCard DISCOVER AMEX Bank PayPal

Pagos seguros

con tu donación nos ayudas a mantener vivo este proyecto

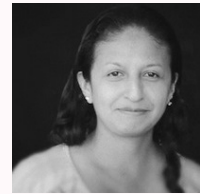
De estudiante a programador

EDUCACIÓN

Por **Yecely Díaz (@silvercorp)**

Yecely es Maestra en Inteligencia Artificial y Desarrolladora Web. Aficionada de la Tecnología, Videojuegos y la Web. Ansiosa de aprender, contribuir y programar todas sus ideas. Ama lo que hace y el código es su vida.

<http://silvercorp.wordpress.com/>



Hace algunas semanas recibí varios correos con temas similares; puedes llamarlo coincidencia o que se acercó fin de semestre para aquellos que estudian, lo cuál me recordó aquella etapa en la que luchaba contra los algoritmos y clases aburridas, donde me preguntaba constantemente: - “¿esto será para mí?”. Tal vez esto, te empiece a sonar familiar...

Algunas personas me han preguntado: - “¿qué necesito para ser un buen programador? ¿en qué momento se volverá todo más sencillo?”. Solo puedo decirles que la programación puede ser un proceso largo pero que cuando logras comprender la lógica del desarrollo, un mundo nuevo llega a ti. Solo puedo decirte: “no desesperes; se paciente; lee; aprende y no te presiones”. Todos tenemos un proceso de aprendizaje diferente y para algunas personas puede ser sencillo pero para otros puede requerir mayor esfuerzo. Solo no olvides que **esta carrera es para aquel que persevera y no se da por vencido.**

“Todo es muy difícil antes de ser sencillo” -Thomas Fuller

La ventaja es que actualmente existen recursos en la red para todos los gustos, es decir desde videotutoriales hasta foros, comunidades, libros electrónicos y revistas, entre otros recursos. Tu decides cómo y dónde te sientes más cómodo para aprender.

Como estos sitios hay muchos más; éstos, son aquellos en los que he aprendido temas de interés y que considero de importancia.

Aprendiendo en línea: Si eres una persona autodidacta, te dejo la lista de algunos enlaces donde podrás encontrar información de diferentes lenguajes de programación así como recursos que te ayudarán a tu auto-aprendizaje.

Codecademy: Es la manera más sencilla que he encontrado de aprender lenguajes como jQuery, HTML, CSS, Python, Ruby, entre otros. Con una consola interactiva e instrucciones paso a paso se te explican cada uno de los lenguajes antes mencionados.

<http://codecademy.com>

Coderwall: Encontraras *tips* de diversas tecnologías, explicados de manera sencilla y concreta por desarrolladores experimentados.

<http://coderwall.com>

Coursera: La Universidad de Stanford ofrece clases totalmente gratuitas por los mejores académicos, con un plan de estudios bien planteado y apoyado de videos, ejercicios y exámenes. Podrás “asistir” a clases como Base de Datos, Introducción a la Inteligencia Artificial, Análisis de Datos, Criptografía, entre otros. Ten en cuenta que se requiere un alto conocimiento del idioma inglés, para poder obtener un mejor provecho de los cursos que tomes. <https://www.coursera.org/courses>

P2P University: Con un enfoque de aprendizaje colectivo, sin restricciones y gratuito podrás aprender diversos temas no únicamente de programación. Cabe destacar que tendrás retroalimentación constructiva de tus avances. <https://p2pu.org/es>



The first Daffodil of Spring

por Susie Oviatt

<http://www.diku.dk/hjemmesider/studerende/thorsbro/SusieOviatt.html>

Para publicar tu mensaje en la Zona U!, envíanos un e-mail contacto@hdmagazine.org, indicando ZonaU! En el asunto del mensaje.

Mensajes de Nuestros Lectores

Edison Moreno (Ecuador):

Estimadas, vi su revista ELVIS #3 y es muy interesante lo que publican, no se si sirva de ayuda; pero se puede usar los feeds para informar cuando salga una revista, puede ser por ejemplo que en un blog se coloque la salida de la revista. y esto llegará al reader que tengan las personas, en mi caso uso el google reader y es muy bueno.

Respuesta:

Hola Edison! Muchas gracias por tu mensaje =) El problema que tenemos en la actualidad, es que somos pocos y estamos trabajando al máximo. Ya no nos queda espacio de tiempo para poder asumir una responsabilidad más. Es por eso, que estamos necesitando del aporte voluntario de la comunidad (ya sea en dinero o trabajo), como para poder cumplir con todo lo que nos piden. No podemos con todo! Pero igual mil gracias por la sugerencia. Y si quieres sumarte a colaborar de forma voluntaria, eres bienvenido! =)

Anónimo (Perú):

Hola, soy un estudiante de Ingeniería de Sistemas. Recién estoy leyendo sus revistas y me parecen muy interesantes. Me encantó la vez que incluyeron sobre la herramienta Pselnt para los que somos nuevos en la programación. Sería útil para los novatos como yo que incluyan una sección en la revista en donde traten temas no tan complicados ni tan avanzados XD . Sigamos así con la revista, hacen un buen trabajo. Perú

Respuesta:

Gracias, de hecho hay algunos artículos que han sido orientados a novatos pero tomaremos en cuenta tu opinión para ver si podemos implementar una sección especial.

Encuesta vía Twitter:

¿Te parecen muy avanzados los artículos publicados hasta ahora? (Responder en: <http://bit.ly/X3stTK>)

12 Respuestas hasta el 23/02/2013:

@edvm nop, están buenos, pero no creo sean avanzados (ahora, no son introductorios ;-)

@jackgris2 Yo creo que están bien ;) Me gustaron los de Arch, yo que uso esa distro podrían agregar algún otro :D

@the_691h están perfectos así como están, muy básicos sería redundar con el material que hay por toda la red

@jatskaneda me parecen geniales además explican muy bien me gustan como están así exitos saludos desde Ccs Vzla

@leonciokof creo que los artículos cada vez se están poniendo mejores

@RootWeiller no, el nivel está muy bien, la metodología para explicar es muy clara.

@DiegoUG Nop, para mí están bien, puede ser que muchos lectores son windows o tienen poca experiencia en linux.

@gacanepa Sí, son avanzados desde mi punto de vista. Podrían incluir algunos básicos junto con esos al mismo tiempo)

@icaroperseo no estaría mal algo más "básico"

@mr_macun no, son justo lo que necesito hay mucho material básico dando vueltas

@josemizamora Para mí sí. Pero en vez de pedirlos más básicos, prefiero avanzar yo ;-)

@magooracss suena mal decir que no? una revista de llena de artículos básicos no necesita el nivel del staff actual.

Anuncios: "Gente que busca gente"

Franco "Bagnado" (ex-conductor de programa de TV de la Rep. Argentina): Vengo a publicar un "llamado a la solidaridad". Un joven de la ciudad de Posadas (Misiones, Argentina), reclama en Twitter, encontrar a "Mujer Linuxera" (?) bajo el hashtag #YoQuieroUnaMujerLinuxera. Quien posea información, contactar con el interesado @javier25arg. Muchas gracias, Franco (el mentor de Guillermo "Arduino", el que encontró a Karinaaaaaa!!!! <http://youtu.be/9X1k2CGjDc>)



safe creative



1 302234 649123
INFO ABOUT RIGHTS

¡GRACIAS POR LEERNOS!