AÑO ---------- 0
NÚMERO -------- 6
FECHA: 2013-04-22

# #6

"FARAÓN"

# 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 GLAMP & Agile Coach |
| Indira Burga | Ingeniera de Sistemas |
| María José Montes Díaz | Técnica en Informática de Gestión |
| Milagros Infante Montero | Est. Ingeniería de Sistemas |
| Sergio Infante Montero | Ingeniero de Software |

# HD Hackers & DEVELOPERS #6

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:



Y LAS SECCIONES DE SIEMPRE:

# 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

**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

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

**Sergio Infante Montero**
Ingeniero de Software
neosergio.net

## 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

Web Oficial: www.hdmagazine.org
Cuenta Twitter Oficial: @HackDevMagazine

# Distribuyendo tus aplicaciones Python en PyPI

**Si desarrollaste un módulo o aplicación Python, distribuirlo en el *Python Package Index* (PyPI) y dejarlo disponible a solo un "pip install" de cualquier usuario, es una tarea sumamente sencilla que en unos pocos minutos puedes aprender. Entérate cómo distribuirlas leyendo este artículo.**

*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**

El *Python Package Index*, más conocido por sus siglas **PyPI**, es el repositorio oficial de aplicaciones desarrolladas en Python, que permite a programadores de todo el mundo distribuir sus creaciones de forma eficiente y muy simple.

Las aplicaciones disponibles en http://pypi.python.org, pueden ser instaladas por los usuarios, mediante un simple `pip install nombre-del-paquete` y para que puedas distribuir tus aplicaciones a través de PyPI, solo es necesario prepararlas seguiendo unos pocos pasos como veremos a continuación.

## Preparar tu aplicación

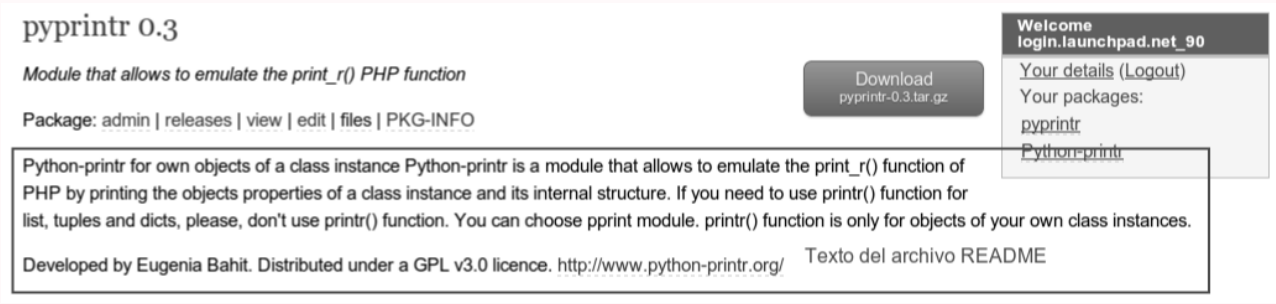
Lo más importante es contar con una carpeta para alojar los módulos y paquetes de la aplicación. Esta carpeta, contará con algunos archivos estándar para su distribución. Una estructura de archivos y directorios apropiada, podría ser como la siguiente:

```
MiAplicacion/            Directorio de la aplicación
|_ paquete_1/            Paquete (parte de la app)
|  |_ __init__.py
|  |_ …
|_ paquete_2/
|  |_ __init__.py
|  |_ …
|_ docs/                 Documentación
|  |_ …
|_ tests/                Test Unitarios
|  |_ …
|_ README.txt            Descripción de la aplicación
|_ LICENSE.txt           Copia de la licencia http://www.gnu.org/licenses/gpl.txt
|_ MANIFEST.in           Lista de archivos que formarán parte del paquete
|_ setup.py              Archivo de instalación
```

### El archivo README.txt

En principio, aclarar que la extensión `.txt` se utiliza en los paquetes para otorgar compatibilidad con sistemas operativos privativos no basados en *nix, pero en sistemas GNU/Linux, dicha extensión, carece de significado. Sin embargo, es un estándar a la hora de distribuir paquetes Python a través de PyPI.

En este archivo se puede colocar cualquier información relativa a la aplicación, preguntas frecuentes, modos de uso e implementación, etc. La misma será aquella que PyPI, muestre en la página del paquete:




Este archivo, así como la documentación que se incluya en docs/*.txt puede escribirse aplicando el formato *reST.*

Una guía rápida de *reST* puede obtenerse en:

http://docutils.sourceforge.net/docs/user/rst/quickstart.html

### El archivo LICENSE.txt

Básicamente se trata de una copia del texto de la licencia con la que se distribuye el Software o paquete. Para distribuir el paquete como Software Libre, recomiendo utilizar la licencia GPL en su versión 3.0 cuyo texto completo puede obtenerse en http://www.gnu.org/licenses/gpl.txt

Recordad que la licencia **GPL v 3.0**, es la recomendada por la ***Free Software Foundation***[1] para la distribución de **Software Libre**, sea éste de uso comercial o no.

### El archivo MANIFEST.in

Éste, es el que almacena el listado de archivos que componen el paquete. El listado de archivos puede colocarse "a mano" (es decir, archivo por archivo) o se pueden utilizar instrucciones como `include` *expresión*.

Para un paquete con la siguiente estructura:

```
PythonPrintr/
|_ printr/
|  |_ __init__.py    -- módulo que contiene la función printr()
|_ docs/
|_ tests/
|_ README.txt
|_ LICENSE.txt
|_ MANIFEST.in
|_ setup.py
```

El archivo `MANIFEST.in` podría verse como el siguiente:

```
include docs/*.txt
include *.txt
include printr/*.py
```

Algunas instrucciones frecuentes que podrían indicarse en un archivo `MANIFEST.in`, son:

```
include
Archivos a incluir en el paquete

exclude
Archivos a ignorar en el paquete

recursive-include
Incluye archivos de forma recursiva

recursive-exclude
Ignora archivos de forma recursiva
```

Otras instrucciones como `global-include`, `global-exclude`, `graft` y `prune` también se encuentran disponibles.

1 http://www.fsf.org/

### El archivo setup.py

Este es el archivo clave. Contendrá toda la información sobre el paquete que deseas distribuir.

Un ejemplo de formato de archivo `setup.py` se verá como el siguiente:

```
from distutils.core import setup


setup(
    name="PythonPrintr",                            # Nombre de la app (CamelCase)
    version="0.3",                                  # Versión
    author="Eugenia Bahit",                         # Nombre del Autor
    author_email="ebahit@member.fsf.org",           # E-mail del autor
    packages=["printr"],                            # Paquetes a incluir
    url="http://www.python-printr.org",             # URL de la app
    license="GPL v3.0",                             # Licencia

    # Descripción corta
    description="Module that allows to emulate the print_r() PHP function",

    long_description=open('README.txt').read(),  # Descripción larga

    # Selección de categorías que pueden encontrarse en:
    # https://pypi.python.org/pypi?:action=list_classifiers
    classifiers=[
        "Development Status :: 5 - Production/Stable",
        "Environment :: Console",
        "Intended Audience :: Developers",
        "License :: OSI Approved :: GNU General Public License v3 (GPLv3)",
        "Natural Language :: English",
        "Operating System :: OS Independent",
        "Programming Language :: Python :: 2.7",
        "Programming Language :: Python :: 2 :: Only",
        "Topic :: Software Development",
        "Topic :: Software Development :: Libraries :: Python Modules"
    ]
)
```

Cada uno de los parámetros que se pasan a la función `setup()`, son opcionales. Sin embargo, los tres primeros (`name`, `version` y `author`) siempre deberían estar presentes.

## Distribuyendo la aplicación

Ya hemos preparado todos los archivos. Es hora de distribuir la aplicación. Para ello, lo primero que haremos es registrar nuestro proyecto en el PyPI. Para ello, desde la terminal, navegamos hasta el directorio de la aplicación y ejecutamos:

```
python setup.py register
```

La primera vez que registremos nuestro proyecto, antes de completar la acción nos

preguntará si deseamos crear una cuenta o utilizar una existente, mostrando el siguiente menú:

```
running register
We need to know who you are, so please choose either:
    1. use your existing login,
    2. register as a new user,
    3. have the server generate a new password for you (and email it to you), or
    4. quit
Your selection [default 1]:
```

Si no dispones de una cuenta en PyPI, la opción 2 será la más apropiada. Eligiendo dicha opción, te pedirá ingresar un e-mail, elegir y confirmar una contraseña alfanumérica que contenga al menos una letra mayúscula.

Una vez aportes dichos datos, te preguntará si deseas guardar los mismos para que no te los pida nuevamente en el futuro.

**ADVERTENCIA:** La contraseña será almacenada en texto plano. Utilizar esta función con suma precaución.

En caso de aceptar esta opción, tus datos serán guardados en el archivo **`~/.pypirc`** con el siguiente formato:

```
[distutils]
index-servers =
    pypi

[pypi]
username:tu_usuario
password:tu_clave
```

Este archivo, es posible crearlo de forma manual.

Finalizado el registro de la cuenta, `distutil` continuará registrando el proyecto:

```
running register
running check
Registering PythonPrintr to http://pypi.python.org/pypi
Server response (200): OK
```

Una vez finalizado, crearemos el *tarball* de distribución y lo subiremos a PyPI en un solo paso:

```
python setup.py sdist --format=gztar upload
```

Al finalizar, un directorio `dist` será creado en la carpeta de la aplicación incluyendo el `.tar.gz`:

```
eugenia@cococha-gnucita:~/webprojects/PythonPrintr$ tree
.
├── dist
│   └── pyprintr-0.3.tar.gz
├── docs
│   └── howto.txt
├── printr
│   └── __init__.py
├── tests
├── LICENSE.txt
├── MANIFEST        -- archivo creado por el sdist
├── MANIFEST.in
├── README.txt
└── setup.py
```

Y esto, es todo lo que necesitamos para distribuir nuestra aplicación.

## Instar tu aplicación

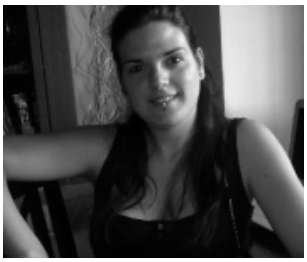
Para visualizar tu aplicación en el repositorio PyPI, deberás ingresar en: `https://pypi.python.org/pypi/NombreDeTuApp`. Luego, para que otros usuarios puedan instalarla, deberán ejecutar:

```
sudo pip install nombre-de-tu-app
```

# Manual de Perl (Parte IV)

**En esta entrega os hablaré del ámbito léxico y espacio de nombres de las variables. Además, veremos cómo se crean subrutinas, así como el paso de parámetros por valor y referencia.**

*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://blog.archninfa.org/

**Redes sociales:**
Twitter: **@MMontesDiaz**

Ya hemos visto las estructuras básicas y cómo se definen y utilizan los diferentes tipos de datos en Perl. Ahora veremos cómo se definen y utilizan las subrutinas (también llamadas funciones o procedimientos). Pero antes de esto, veamos cómo funciona el ámbito léxico de las variables en este lenguaje.

Disponemos de varios métodos para declarar las variables:

```
#Variable global, independientemente de dónde se defina.
$variable = 0;

#Es una variable con ámbito en el bloque en el que se creó.
#Si hay bloques en el interior del mismo, sería accesible también desde ellos.
my $variable = 0;

#Guarda el valor de la variable global y lo sustituye por el nuevo valor que le
#demos. Una vez terminada la ejecución del bloque, se restaura su valor.
#Sólo es aplicable a variables globales.
local $variable = 0;
```

Para evitar errores, una buena práctica es utilizar el pragma strict[2]. Entre otras cosas, fuerza a declarar las variables con **my**. Para declarar variables globales, debemos utilizar **our**. Veamos con un ejemplo:

2 http://perldoc.perl.org/strict.html

```
#!/usr/bin/perl
use strict;

#Declaramos las variables globales
our $global_de_paquete;

#Inicializamos las variables
$global_de_paquete = 10;
my $var_bloque = 10;

#Si descomentamos, da error, no se puede utilizar así sin declarar con our
#$global = 10;

#Sin our, estará disponible pero debemos utilizar siempre el nombre completo:
$main::global = 10;

#Nos da error, ya que $global no ha sido declarada con our. Para utilizarla,
#con su nombre completo
#local $global = 10;
local $main::global = 10;
```

En Perl podemos agrupar las declaraciones de variables mediante los paquetes. Lo que se hace es que se asocian las variables al nombre del paquete que se definió anteriormente. Se utiliza el paquete **main** de forma implícita. Esto quiere decir que todas las variables declaradas en el ejemplo anterior pertenecerían al paquete **main**. A esto es lo que se conoce como espacio de nombres.

En cuanto al ámbito, debemos tener en cuenta que las variables globales son accesibles desde cualquier parte del código, siempre que utilicemos su nombre completo.

```
#!/usr/bin/perl
# Sin utilización de strict;

package global;

#Inicializamos las variables
$global_de_paquete = 10;

package main;

#Es una operación de asignación, no de declaración, por el anterior uso de our
$global_de_paquete = 9;

print $global_de_paquete;
#imprime: 9

package global;
print $global_de_paquete;
#imprime: 10;
```

Cómo podemos ver con el ejemplo anterior, el hecho de no utilizar **`strict`** puede provocar errores. Cuando accedemos a `$global_de_paquete`, por defecto Perl, en caso de no existir, la declara como variable global de ese paquete. Si nuestra intención era esa, no pasa nada, pero si en realidad queríamos acceder a la variable global...

En el caso de haber utilizado **`strict`**, el *script* anterior habría dado un error. Con `strict`

estamos obligados a declarar las variables globales con **our** para que no necesitemos utilizar su nombre completo.

Las variables definidas con **my** son variables locales al bloque en que son declaradas. Por supuesto, si hay bloques dentro de éste, se podrá acceder a ella desde esos bloques.

## Subrutinas

A la hora de escribir programas, es necesario la utilización de subrutinas para poder aplicar la técnica de "divide y vencerás". Una subrutina (también llamadas procedimientos o funciones) es un subprograma que se referencia con un nombre y puede ser llamado desde otra parte del programa principal las veces que necesitemos.

Su sintaxis en Perl es:

```
sub nombre_subrutina {
   bloque_a_ejecutar;
   [return expresión;]
}
```

Al llamar a una subrutina, podemos enviarle parámetros. Éstos, a diferencia de otros lenguajes, son dinámicos y no hemos de especificarlos en la definición.

En la subrutina, la variable especial `@_` es quien contendrá la lista de parámetros que le han sido pasados. Para acceder a cada uno de ellos, podemos utilizar `$_[0], $_[1],..., $_[n]`

Por defecto, las subrutinas devuelven el último valor de la última expresión que haya en la misma, si no se especifica nada con **return**.

Disponemos de varios métodos para llamar a una subrutina:

```
#Llama a la subrutina con los actuales valores de @_, equivale a & subrutina (@_);
& subrutina;

# Si no se  pasan parámetros en los paréntesis, se envía una lista nula.
& subrutina ([parámetro1,parámetro2...]);
subrutina ([parámetro1,parámetro2...]);

#Equivale a subrutina ();
subrutina [parámetro1,parámetro2...];
```

```
#!/usr/bin/perl
use strict;

#No hay prototipo, podemos pasar cualquier número de parámetros.
sub factorial {
        my ($n) = @_;
        my $resul = 0;
```

```
        return $result if ( $n < 0 );
        $result = 1;
        for (my $i=1;$i <= $n; $i++) { $resul*=$i; }
        return $resul;
}

#También podríamos definirla de forma recursiva
#Hay prototipo, forzamos a que se pase un escalar.
sub factorial2 ($){
        my ($num) = @_;

        return 0 if ( $num <0 );
        return 1 if ( $num == 0 || $num == 1 );
        return $num * factorial ($num -1);
}


for (my $i=0;$i<5;$i++) {
   print "\n $i! =", &factorial($i);
}
```

Para poder modificar el valor de los parámetros, es decir, paso por referencia, debemos definir las variables que contendrán los argumentos precedidas de un asterisco y con `local`:

```
sub subrutina {
    local (*arg1,*arg2) = @_;
    …
}
```

Un ejemplo con una subrutina que intercambia el valor de los argumentos:

```
#!/usr/bin/perl
use strict;
my ($x,$y)=(1,2);

#Intercambia el valor de dos variables
#Especificamos que se han de pasar dos variables por referencia
sub  swap (**){
        local (*x, *y) = @_;
        my $aux ;

        $aux=$x;
        $x=$y;
        $y=$aux;
        return @_;
}

#Este ejemplo también funciona, llamaremos a la función con la variable,
#no con la referencia.
sub swap2 (**){
        my $aux;

        $aux=$_[0];
        $_[0]=$_[1];
        $_[1]=$aux;
```

```
        retrun @_;
}

swap(*x, *y);
print "\nNum1: $x, Num2: $y";
#Imprime: Num1: 2, Num2: 1"

#volvemos a intercambiar las variables, aunque no hemos pasado
#las variables por referencia.
swap2($x,$y);
print "\nNum1: $x, num2: $y";
```

Cómo hemos podido ver, en la subrutina `swap2`, aunque no pasamos las variables por referencia, se han intercambiado sus valores. En Perl, en realidad, todos los parámetros se pasan por referencia.

Para prevenir errores, es conveniente no utilizar directamente las variables por defecto ( **`$_[]`** ). Cuando necesitemos utilizar parámetros, es buena idea utilizar siempre variables locales. Para los parámetros pasados por referencia, las definiremos con **`local`** y, para los parámetros por valor, lo haremos con **`my`**.

**Enlaces de interés:**

http://www.perl.com/

http://www.cpan.org/

# Equipos Ágiles: Parte II

**Es muy útil saber como es un típico equipo ágil, como puedes formar uno y que necesitas saber antes de empezar con el trabajo.**

*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

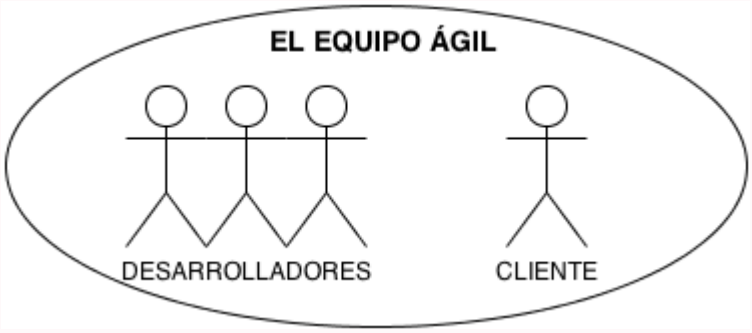
Twitter: **@neosergio**

Las metodologías ágiles como tal no tienen roles formales, sólo se tiene algo realmente claro: hay personas que tienen necesidades, de las cuáles se obtienen los requisitos de lo que se construirá (clientes) y personas que pueden construir la solución a esas necesidades (desarrolladores).

> *En términos simples tenemos a los que deciden **qué se va a construir** y a los que deciden **cómo se construirá**.*

Por lo tanto si aparece la pregunta: ¿dónde están los programadores, *testers* y analistas?, la respuesta corta es: siguen estando ahí.

El agilismo no busca descubrir quiénes interpretan los roles, ni el rol mágico que todo lo puede sino, se enfoca en que los roles sean realmente los apropiados para la solución.




# El cliente ágil

*El cliente ágil es la fuente principal de la verdad*

Todos los requerimientos vienen del cliente ágil. Él conoce el flujo de la información, es la fuente de las necesidades por resolver y como es de esperarse es para quien se está construyendo el software.

Está familiarizado con el negocio y es al que le importa de verdad que el software cumpla con su función; él provee los lineamientos al equipo sobre el giro del negocio, resuelve preguntas y obviamente proporciona retroalimentación de información, a la vez que define prioridades desde el punto de vista del negocio y decide qué se construye y cuándo.

Todas estas responsabilidades del cliente no van solas; es un proceso de colaboración con el equipo de desarrollo, que entiende que hay razones técnicas para tener que priorizar algunas tareas y funcionalidades antes que otras.

Estos clientes tienen el trabajo de decidir los tiempos de entrega y proporcionar el financiamiento para empezar con el proyecto.

Idealmente el cliente debe trabajar a tiempo completo con el equipo de desarrollo para hacer que las cosas funcionen, por ejemplo en las primeras versiones de la metodología XP, se sugería que el cliente ocupe el mismo espacio físico que el equipo.

> En Scrum al cliente se le conoce como el dueño del producto.

Sin embargo es posible que no se pueda obtener esa exclusividad a tiempo completo; sin embargo a pesar de ese posible contratiempo, el compromiso que se debe tener con el cliente debe ser fuerte para que el equipo sea capaz de tomar decisiones con el consentimiento y el conocimiento completo del cliente para el éxito del proyecto.

*Un cliente involucrado significa un proyecto exitoso.*

# El equipo de desarrollo[3]

Recordando que alguna vez hemos visto o leído estos roles: analistas de negocios, gestores o gerentes de proyecto, administradores de bases de datos, evaluadores, diseñadores, programadores, documentadores, entre otros que siguen estando presente. Pues el equipo de desarrollo ágil es básicamente un grupo multidisciplinario de personas que hacen realidad los requerimientos de los clientes, solucionan una necesidad, sin importar tanto que rol estricto cumplan.

En la teoría y lo ideal se dice que son auto-organizados y hasta en cierto grado lo son en la práctica, sin embargo hay algunos roles que necesitan ser clarificados para que no se vuelva confuso y evitar que se convierta en un equipo de irresponsables.

### El analista ágil

Cuando un requerimiento llega para ser desarrollado, alguien tiene que asumir la labor de involucrarse y comunicarse con el cliente, para asegurarse de haber entendido la necesidad a resolver. Esa labor es la del analista ágil.

Ayuda al cliente a escribir historias de usuario, crea bosquejos, prototipos y usa todas sus habilidades para ayudar a entender la esencia de la historia de usuario.

Su labor es muy importante, entiende el lenguaje del cliente y el lenguaje del equipo técnico.

### El programador ágil

Este es el rol que escribe código y se toma en serio su trabajo, porque lo hace siempre pensando en la mejora continua y en la mejora de la calidad del producto, automatiza pruebas y hace uso de las buenas prácticas de desarrollo, esta constantemente haciendo mejoras al software.

Trabaja también muy cerca del cliente y de todos los demás en el equipo para asegurarse de que todo funcione, entregando constantemente juntos, software funcional que resuelve necesidades del cliente.

### El evaluador ágil

Los evaluadores ágiles se hacen cargo de prevenir errores en el producto que se va a entregar. Es por ello que participan desde el principio del proyecto, asegurando el éxito de la relación entre la historia de usuario y la funcionalidad entregada.

Están por todo el proyecto. Esto se debe a que en un proyecto ágil todo necesita probarse y además, ayuda también capturando requerimientos de los clientes a través de pruebas.

---

3 NdR: por favor, nótese que los roles indicados por el autor en este apartado, son usualmente visualizados al trabajar con metodologías como Kanban o en organizaciones ágiles más ligeras que no se encuentran arraigadas a una metodología puntual. Sin embargo, en la utilización de metodologías ágiles menos ligeras como eXtreme Programming, si bien no se especifican ni se definen roles, podrían resultar contrarios a algunos de sus principios mientras que en otras metodologías como Scrum, directamente representarían roles opuestos a los propuestos por la misma.

Los evaluadores ágiles ayudan a los programadores a automatizar pruebas y encontrar defectos en el corto plazo, para evitar que se generen grandes gastos de mantenimiento y soporte.

Tiene la visión completa de lo que debe ser evaluado (pruebas de carga, esfuerzo, escalabilidad, usabilidad y todo lo que pueda ser evaluado).

### El gestor de proyecto ágil

Este rol conoce el camino para que el equipo tenga éxito, se encarga de remover cualquier impedimento o problema que dificulte el trabajo del equipo.

Está constantemente planeando y replaneando, ajustando las cosas cuando sean necesarias, presenta los reportes a los que financian el proyecto, refuerza las relaciones entre las personas del equipo y protege al equipo cuando es necesario.

Un buen gestor de proyecto, no le dice al equipo qué hacer ya que no es necesario hacerlo y además crea el ambiente para que el equipo sea independiente y continúe entregando valor al cliente.

### El diseñador UX ágil

La experiencia del usuario es un punto fundamental para el éxito de un software; la labor de este diseñador es estar enfocado en crear interfaces funcionales, usables y orientadas 100% al usuario y su contexto.

Debe encontrar y usar todos los medios posibles para entender al usuario del software, y colaborar con el resto del equipo para encontrar la mejor interfaz.

Afortunadamente, muchas de las prácticas usadas por los expertos en usabilidad vienen derivadas del espíritu ágil, como enfocarse en el valor, la retroalimentación rápida y buscar mejorar constantemente.

Los diseñadores UX no temen diseñar interfaces de manera incremental e iterativa; tener a un diseñador UX en un equipo es bastante importante, ya que ayuda a que fluyan las ideas constantemente, siempre pensando en el usuario final.

### Los demás

Existen roles como los administradores de bases de datos, documentadores, administradores de sistemas, arquitectos de software, coordinadores, entrenadores, entre otros. Ellos también forman parte del equipo y deben ser tratados por igual como todos los demás.

> Scrum tiene un rol llamado Scrum Master que es una especie de entrenador y un super gestor de proyecto todo en uno

Los equipos experimentados no necesitan ya de entrenadores dedicados al equipo,

pero los nuevos equipos definitivamente encontrarán beneficio en tener a uno.

> *En un equipo ágil es normal que una persona pueda tener varios roles, ya que no son excluyentes entre si.*

## Consejos para formar tu equipo ágil

- Busca a los generalistas, aquellos programadores que no tengan miedo a tocar cualquier parte del proyecto. No tendrás problemas con ellos ya que tendrán una visión global sobre el proyecto y no tendrán problemas con interpretar diversos roles.
- Personas que se sientan cómodas con el cambio, aquellos que no tienen problemas en aprender nuevas tecnologías y se adaptan rápidamente a una. Un proyecto cambia todo el tiempo; ver al cambio no como un problema, sino como una oportunidad.
- Personas que sepan dejar sus egos afuera, por el bien del proyecto; personas que puedan entender y aprovechar el conocimiento de otros y compartir información; personas que disfruten de aprender y crecer en equipo.

> ***Se aprende a hacer un equipo ágil, participando de uno o intentando formar uno.*** *La teoría es insuficiente; la práctica clarificará los conceptos.*

## GNU/Linux para programadores:

# Permiso denegado

***Change Mode*, más conocido por su abreviatura `chmod`, es el comando de GNU/Linux que permite modificar los permisos de acceso tanto a archivos como a directorios. En más de una oportunidad te habrás encontrado ejecutando el comando `chmod` seguido de 777 como argumento para solucionar el error "Permiso denegado". Sin embargo, la mayoría de las veces, esa, no siempre es la solución correcta.**

*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**

Cada vez que un alumno ejecuta un `ls -l` y veo que cambió los permisos de todo un directorio Web a 777, en mi cabeza aparece la imagen de *Bruce Dikinson* gritando *"Six, six, six, the number of the beast"*. Claro que 666 no es lo mismo que 777, pero **para el sistema de archivos en GNU/Linux el 777 es** -definitivamente-, **el número de la bestia** ¡créeme!

Nada más frecuente para un programador, que ejecutar su aplicación Web en el navegador y ver el error "Permiso denegado". Pero el mismo, no necesariamente se produce porque un archivo o directorio tenga permisos diferentes a 777. Sin embargo, la primera "solución" que el programador suele encontrar, es cambiar los permisos asignando algo tan "endemoniadamente" peligroso como 777.

Asignar a un archivo o directorio permisos 777, significa asignar permisos de lectura, escritura y ejecución sobre ese archivo o directorio, para cualquier usuario, sea o no el propietario. Pero estos números, no son números al azar, ni mucho menos se necesita andar memorizando números de 3 cifras para entender estos permisos en GNU/Linux. Simplemente, se trata de comprender cómo funciona el sistema de permisos.

Cuando hacemos un listado de archivos y directorios, a la izquierda de cada archivo

(como primera columna) vemos una sucesión de 10 de caracteres:

```
-rw-rw-r--  1 eugenia eugenia  18K mar 29 12:54 Agenda 2013.ods
drwxrwxr-x  2 eugenia eugenia 4,0K ene  5 18:49 certificados
lrwxrwxrwx  1 root root         45 nov  9 18:49 projects -> /home/eugenia/projects
```

A la vez, la tercera columna, muestra el nombre del propietario del archivo (generalmente, usuario que creó el archivo o directorio, o usuario que ha sido asignado como "dueño" por el creador del archivo):

```
-rw-rw-r--  1 eugenia eugenia  18K mar 29 12:54 Agenda 2013.ods
drwxrwxr-x  2 eugenia eugenia 4,0K ene  5 18:49 certificados
lrwxrwxrwx  1 root root         45 nov  9 18:49 projects -> /home/eugenia/projects
```

La cuarta columna indica el grupo al cuál el archivo o directorio pertenece:

```
-rw-rw-r--  1 eugenia eugenia  18K mar 29 12:54 Agenda 2013.ods
drwxrwxr-x  2 eugenia eugenia 4,0K ene  5 18:49 certificados
lrwxrwxrwx  1 root root         45 nov  9 18:49 projects -> /home/eugenia/projects
```

Volviendo a la primera columna (la que muestra la sucesión de 10 caracteres), se subdivide (aunque imaginariamente) en 4 bloques. El primero, de un solo *caracter* y los 3 restantes, de 3 caracteres cada uno:

```
-  rw-  rw-  r--  1 eugenia eugenia  18K mar 29 12:54 Agenda 2013.ods
d  rwx  rwx  r-x  2 eugenia eugenia 4,0K ene  5 18:49 certificados
l  rwx  rwx  rwx  1 root root         45 nov  9 18:49 projects -> /home/eugen...
```

El **primer bloque** de un *caracter*, indica simplemente el **tipo** de elemento del cuál se trata:

```
-          Archivo
d          Directorio
l          Enlace simbólico
```

Los 3 caracteres de cada uno de los **bloques restantes**, indican los **permisos** de acceso:

```
r          (read)      Lectura
w          (write)     Escritura
x          (eXecution) Ejecución
-          sin permiso
```

Esos bloques de permisos, corresponden al propietario, al grupo y a cualquier otro usuario respectivamente:

```
   propietario  grupo  otros
-  rw-          rw-    r--  1 eugenia eugenia  18K mar 29 12:54 Agenda 2013.ods
d  rwx          rwx    r-x  2 eugenia eugenia 4,0K ene  5 18:49 certificados
l  rwx          rwx    rwx  1 root root         45 nov  9 18:49 projects -> /ho...
```

Si tomamos el primer archivo, estamos diciendo que:

```
rw- El propietario tiene permisos de lectura y escritura pero no de ejecución
rw- El grupo tiene permisos de lectura y escritura pero no de ejecución
r-- Otros usuarios solo tienen permisos de lectura
```

En cambio, el tercer archivo (enlace simbólico) tiene permisos de lectura, escritura y ejecución para el propietario, el grupo y para cualquier otro usuario:

```
Propietario: rwx (read, write, execute)
Grupo:       rwx (read, write, execute)
Otros:       rwx (read, write, execute)
```

Las letras que representan cada uno de los permisos, se encuentran asociadas a un número:

```
r   read     lectura    4
w   write    escritura  2
x   execute  ejecución  1
```

Regla para memorizar la asociación de números y letras/permisos: **Cuanto mayor es el número, menor es el permiso.**

Cada bloque de 3 letras, es representado por la suma de los números de cada letra. Es decir que sumando los permisos de cada bloque, se obtiene el "número final" de permiso:

```
r--r--r-- [4]     [4]   [4]          444
-w--w--w- [2]     [2]   [2]          222
rw-rw-rw- [4+2]   [4+2] [4+2]        666
rw-r--r-- [4+2]   [4]   [4]          644
rwxrw-r-- [4+2+1] [4+2] [4]          764
```

Antes de modificar los permisos de un archivo o directorio, se debe pensar previamente en qué necesidades tendrá cada uno de los bloques. Por ejemplo:

```
Propietario: permisos absolutos                  rwx = 4 + 2 + 1 = 7
Grupo:       permisos de lectura y escritura     rw- = 4 + 2 + 0 = 6
Otros:       permisos de lectura                 r-- = 4 + 0 + 0 = 4
```

Un error frecuente, es asignar permisos absolutos (777) para que un usuario en particular, distinto al propietario, pueda tener un acceso completo al archivo:

```
$ ls -l
-rw-rw-r--  1 eugenia developers  18K mar 29 12:54 archivo.foo

$ chmod 777 archivo.foo
$ ls -l
-rwxrwxrwx  1 eugenia developers  18K mar 29 12:54 archivo.foo
```

Pero en el caso anterior, si solo se quiere que, por ejemplo, el usuario "juan" tenga un acceso completo, se debe cambiar el propietario del archivo y no los permisos:

```
$ ls -l
-rw-rw-r--  1 eugenia developers  18K mar 29 12:54 archivo.foo

$ chown juan archivo.foo
$ ls -l
-rw-rw-r--  1 juan developers  18K mar 29 12:54 archivo.foo
```

De esta forma, el usuario juan por ser propietario del archivo, tendrá acceso para leerlo y escribirlo y el usuario eugenia, por pertenecer al grupo developers, tendrá el mismo acceso al igual que cualquier otro usuario del grupo developers. La alternativa, también podría haber sido incorporar al usuario juan al grupo developers pero ya nos estaríamos yendo del tema.

Si en cambio, solo se quisiera que los usuarios juan y eugenia tuviesen acceso al archivo, pero solo juan pudiese escribir en él, se coloca como propietario a juan y como grupo, uno al que solo pertenezca el usuario eugenia y luego, se modifican los permisos para dicho grupo:

```
$ ls -l
-rw-rw-r--  1 eugenia developers  18K mar 29 12:54 archivo.foo

$ chown juan:grupo_eugenia archivo.foo
$ ls -l
-rw-rw-r--  1 juan grupo_eugenia  18K mar 29 12:54 archivo.foo

# chmod 640 archivo.foo
$ ls -l
-rw-r--r--  1 juan grupo_eugenia  18K mar 29 12:54 archivo.foo
```

Pero el cambio de propiedad, **no debe aplicarse a usuarios como** por ejemplo, el usuario de Apache **www-data**. Cuando a este usuario se le otorga la propiedad de un

archivo, debe entenderse que ***se está otorgando la propiedad a cualquier persona que acceda a dicho archivo mediante Apache***. Por ejemplo, si se tratara del archivo `foo.html` al cual se accede mediante `http://www.example.org/foo.html`, cualquier persona que accediera a dicha URL, será considerada propietaria de ese archivo, ya que el mismo, será ejecutado por el usuario `www-data`.

Siempre hay que tener en cuenta que además del súper usuario, el propietario de un archivo es quien puede realizar cambios sobre éste, como por ejemplo, cambiar la propiedad del mismo o modificar sus permisos de acceso.

Entonces **¿qué hacer si se necesita que el usuario `www-data` tenga un acceso especial a determinados archivos?**

En principio, al usuario `www-data` se lo debe considerar en el tercer grupo de "cualquier otro usuario". Que el usuario `www-data` sea el usuario de Apache, no significa que solo Apache utilizará dicho usuario. Indirectamente, cualquier persona que realice una petición al servidor, lo estará haciendo en nombre del usuario `www-data`.

Luego, hay que analizar con cautela, qué necesidades reales tendrá este usuario. Por defecto, la única necesidad real de los archivos accedidos por el usuario `www-data` es la de lectura. Para los directorios, se suma la de ejecución a fin de que se pueda ingresar en ellos. Por lo tanto, **un directorio web debería tener los siguientes permisos**:

- Propietario: lectura, escritura y ejecución: 4 + 2 + 1 = 7
- Grupo: no son necesarios permisos: 0 (aunque una buena práctica, es asignar los mismos permisos al grupo que al resto de los usuarios)
- Resto de los usuarios (incluye a www-data): lectura y ejecución: 4 + 1 = 5

```
chmod 705 directorio_web # Opción 1: Grupo sin permisos
chmod 755 directorio_web # Opción 2: Grupo mismos permisos que el resto de usuarios
```

Luego, los archivos tendrán los mismos permisos que el directorio, menos el de ejecución para el resto de usuarios:

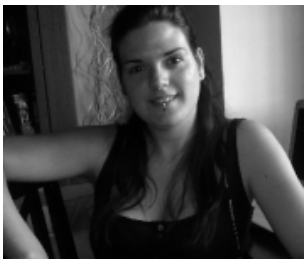
```
chmod 704 archivo_web # Opción 1: Grupo sin permisos
chmod 744 archivo_web # Opción 2: Grupo mismos permisos que el resto de usuarios
```

*Asignando 705 a los directorios y 704 a los archivos, "Permiso Denegado" sería un error que jamás deberías volver a ver.*

# Pásate a GNU/Linux con ArchLinux: NGiNX+PHP+MariaDB

**En esté artículo vamos a instalar un LEMP (Linux+NGiNX+MariaDB+PHP). Para el servidor Web, utilizaremos una técnica de enjaulado, que nos permitirá aumentar la seguridad.**

*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://archninfa.blogspot.com.es/

**Redes sociales:**
Twitter: **@MMontesDiaz**

NGiNX es un servidor HTTP libre, de código abierto, de alto rendimiento y puede ser utilizado para actuar tanto como proxy HTTP reverso como proxy IMAP/POP3.

*Igor Sysoev* comenzó su desarrollo en 2002 para solventar un problema llamado C10K[4]. Es conocido por su alto rendimiento, estabilidad, una configuración sencilla y un consumo bajo de recursos.

Vamos a instalar los paquetes necesarios:

```
# pacman -Syu nginx php-fpm mariadb mariadb-clients
```

## Enjaulando NGiNX

Para añadir seguridad, el servidor web lo vamos a enjaular. Lo primero que necesitamos es crear la estructura de carpetas, por ejemplo `/srv/nginx`:

4 http://www.kegel.com/c10k.html

```
# mkdir -p /srv/nginx/dev
# mkdir -p /srv/nginx/etc/nginx/
# mkdir -p /srv/nginx/usr/{lib,sbin}
# mkdir -p /srv/nginx/usr/share/nginx
# mkdir -p /srv/nginx/var/{log,lib}/nginx
# mkdir -p /srv/nginx/srv/html/
# mkdir -p /srv/nginx/{run,tmp}
# cd /srv/nginx/
# ln -s usr/lib lib
```

Si nuestro sistema es **x86_64**:

```
# ln -s usr/lib lib64
# cd usr
# ln -s lib lib64
```

Debemos crear algunos dispositivos que utiliza NGiNX. Para ello, ejecutaremos lo siguiente:

```
# mknod -m 0666 /srv/nginx/dev/null c 1 3
# mknod -m 0666 /srv/nginx/dev/random c 1 8
# mknod -m 0444 /srv/nginx/dev/urandom c 1 9
```

Ahora debemos copiar los archivos:

```
# cp -r /usr/share/nginx/* /srv/nginx/usr/share/nginx
# cp -r /usr/share/nginx/html/* /srv/nginx/srv/html
# cp /usr/sbin/nginx /srv/nginx/usr/sbin/
# cp -rp /var/lib/nginx /srv/nginx/var/lib/

# touch /srv/nginx/etc/shells
# touch /srv/nginx/run/nginx.pid
```

Después de esto, debemos copiar las bibliotecas necesarias para que pueda ejecutarse NGiNX:

```
# cp /lib/ld-linux* /srv/nginx/lib
# cp $(ldd /usr/sbin/nginx |grep /usr/lib|sed -r 's/(.+) (\/.*) (.*)/\2/')
/srv/nginx/usr/lib
# cp /usr/lib/libnss_* /srv/nginx/usr/lib
# cp -rfvL /etc/
{services,localtime,nsswitch.conf,nscd.conf,protocols,hosts,ld.so.cache,ld.so.conf,
resolv.conf,host.conf,nginx} /srv/nginx/etc
```

Ahora, vamos a crear los usuarios necesarios para poder ejecutar el servidor en el **chroot**:

```
# echo -e  "http:x:33: \nnobody:x:99:" > /srv/nginx/etc/group
# echo -e  "http:x:33:33:http:/:/bin/false \nnobody:x:99:99:nobody:/:/bin/false"
> /srv/nginx/etc/passwd
# echo -e  "http:::\nnobody:::" > /srv/nginx/etc/gshadow
```

En el caso de querer utilizar un puerto hasta el 1024, deberemos ejecutar:

```
# setcap 'cap_net_bind_service=+ep' /srv/nginx/usr/sbin/nginx
```

Ya sólo falta establecer los permisos correspondientes para que pueda ejecutarse y montar los dispositivos:

```
 # chmod +rw /srv/nginx/tmp
 # chmod +rw /srv/nginx/run
 # chown  http:log /srv/nginx/var/log/nginx

 # mount -t tmpfs none /srv/nginx/run -o 'noexec,size=1M'
 # mount -t tmpfs none /srv/nginx/tmp -o 'noexec,size=100M'
```

Una vez hecho esto, vamos a crear la unidad de servicio que se encargará de levantar NGiNX.  Para ello, primero creamos la carpeta para alojar los servicios locales (si no está ya creada):

```
# mkdir -p /usr/local/lib/systemd/system
```

Creamos el archivo `/usr/local/lib/systemd/system/nginx-jail.service` y le añadimos:

```
 [Unit]
 Description=A high performance web server and a reverse proxy server
 After=syslog.target network.target

 [Service]
 Type=forking
 PIDFile=/srv/nginx/run/nginx.pid
 ExecStartPre=/usr/bin/chroot --userspec=http:http /srv/nginx /usr/sbin/nginx -t -q
-g 'pid /run/nginx.pid; daemon on; master_process on;'

 ExecStart=/usr/bin/chroot --userspec=http:http /srv/nginx /usr/sbin/nginx -g
'pid /run/nginx.pid; daemon on; master_process on;'

 ExecReload=/usr/bin/chroot --userspec=http:http /srv/nginx /usr/sbin/nginx -g 'pid
/run/nginx.pid; daemon on; master_process on;' -s reload

 ExecStop=/usr/bin/chroot --userspec=http:http /srv/ngingx /usr/sbin/nginx -g
'pid /run/nginx.pid;' -s quit

 [Install]
```

```
 WantedBy=multi-user.target
```

Pues ya tenemos nuestro servidor web instalado. Iniciamos el servicio:

```
# systemctl start nginx-jail.service
```

Y comprobamos que todo está correcto accediendo a http://localhost.

Para que se inicie en cada arranque:

```
# systemctl enable nginx-jail.service
```

Debemos tener en cuenta que, al instalar así, aunque actualicemos el sistema, no se actualizará la jaula. Deberemos volver a copiar los archivos, tanto del propio NGiNX, como de las bibliotecas.

## Configurando php-fpm

Ya tenemos el servidor web corriendo. Ahora vamos a configurar el `php-fpm`, con el que daremos soporte PHP a NGiNX.

Para ello editamos el archivo `/etc/php/php-fpm.conf` y localizamos las líneas:

```
;listen = 127.0.0.1:9000
listen = /run/php-fpm/php-fpm.sock
```

Y las cambiamos por:

```
;listen = 127.0.0.1:9000
listen = /run/php-fpm/php-fpm.sock
```

Después buscamos la entrada `chroot`, dejándola así:

```
chroot = /srv/nginx/
```

Levantamos el servicio:

```
# systemctl start php-fpm.service
```

Para habilitarlo en el arranque:

```
# systemctl enable php-fpm.service
```

Editamos el archivo de configuración de NGiNX, `/srv/nginx/etc/nginx/nginx.conf` y lo dejamos así:

```
    server {

        ...

        location / {
            root   /srv/html/;
            index  index.html index.htm index.php;
        }

        ...

        location ~ \.php$ {
            root           /srv/html;
            fastcgi_pass   127.0.0.1:9000;
            fastcgi_index  index.php;
            fastcgi_param  PHP_ADMIN_VALUE  open_basedir="/srv/html";
            include        fastcgi.conf;
        }

        ...
    }
```

Para probar que todo va correctamente, creamos un archivo en `/srv/nginx/srv/html`, llamado `index.php` con el siguiente contenido:

```
<?php
  phpinfo();
?>
```

Ahora, tras reiniciar NGiNX, al acceder a http://localhost/index.php, debe mostrarnos la información acerca de nuestra versión de de PHP.

# Configuración de MariaDB

**MariaDB[5] es un servidor de base de datos derivado de MySQL[6], con licencia GPL.** Posee las mismas órdenes, interfaces, APIs y bibliotecas, por lo que su compatibilidad con MySQL® es altísima.

Cuando Oracle® compró Sun Mircrosystems, se creó este *fork* para asegurar que permanecía una versión GPL. Este proyecto está desarrollado por Michael Widenius[7], fundador de MySQL®.

Ya tenemos el soporte PHP y el servidor web corriendo. Vamos a levantar la base de datos MariaDB.

Primero, para que estén disponibles las extensiones de PHP para el acceso a MariaDB, crearemos el archivo `/etc/php/conf.d/mariadb.ini` con el siguiente contenido:

```
extension=pdo_mysql.so
extension=mysqli.so
extension=mysql.so
```

Reiniciamos el servidor web y `php-fpm`:

```
# systemctl restart php-fpm.service nginx-jail.service
```

Ahora debemos configurar la BBDD. Para ello utilizaremos los siguientes comandos:

```
# systemctl start mysqld.service
# mysql_secure_installation
```

Reiniciamos la BBDD y habilitamos su arranque en el inicio:

```
# systemctl restart mysqld.service
# systemctl enable mysqld.service
```

Por supuesto, si queremos, podemos correr la base de datos en una jaula, utilizando un método similar al que hemos empleado para enjaular NGiNX.

Enlaces de interés: https://wiki.archlinux.org/index.php/Nginx

5 https://mariadb.org/
6 http://www.mysql.com/
7 http://es.wikipedia.org/wiki/Michael_Widenius

# *Refactoring:* otra práctica de la Programación eXtrema

**En ediciones anteriores estuvimos hablando de TDD, sus beneficios y forma de implementarlo. En esta edición, nos concentraremos en el *Refactoring*: otra de las prácticas técnicas sugeridas por eXtreme Programming.**

*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**

*Refactoring*, es una de las prácticas técnicas sugeridas por XP[8] que consiste en mejorar el código fuente de una aplicación, sin que dichas modificaciones, afecten el comportamiento externo del sistema.

Cuando se programa con TDD[9] se hace imposible evitar *refactorizar* el código fuente ya que de los propios test deriva esta necesidad. Sin embargo, las técnicas de *Refactoring* pueden implementarse como una buena práctica de programación aunque no se programe con TDD.

Existen diferentes tipos de *refactorizaciones* que pueden ser necesarias implementar al código de nuestra aplicación. Cada tipo, representa una técnica diferente de *refactorización*. Por ejemplo, eliminar código redundante requiere de una técnica diferente a dividir los algoritmos de un método para crear métodos derivados.

8 Extreme Programming
9 Test Driven Development (Desarrollo guiado por pruebas)

## Un problema no es un error...

Antes de continuar, habría que diferenciar el término "problema" de la palabra "error", para no generar confusiones.

El error en sí, es una falla en el código fuente que impide el correcto comportamiento del sistema. Mientras que el problema, puede definirse como "algo que huele mal en el código fuente"[10] pero sin embargo, no impide el correcto funcionamiento de la aplicación.

Los problemas que se pueden presentar en el código fuente de una aplicación, dependen de muchísimos factores y en gran parte de los casos, encuentran una relación directa con el paradigma de programación empleado así como en el lenguaje que se utilice.

Si se intentara abarcar todos los problemas posibles, la lista podría tornarse infinita, tediosa y hasta inútil o incluso confusa. Es por ello, que solo abarcaremos los problemas más frecuentes que puedan considerarse generales con independencia del lenguaje en que se programe. Trataré de incluir ejemplos tanto en Python como en PHP para hacer el tema más general.

## La regla

En el mundo del *Refactoring*, haciendo una analogía con el béisbol, suele utilizarse la regla ***"Tres Strikes[11] y ¡refactoriza!"***. Esta regla puede traducirse como:

> *La primera vez que hagas algo, solo hazlo. La segunda vez que hagas algo similar, notarás que estás duplicando código, pero lo harás de todas formas. La tercera vez que te enfrentes al mismo caso, refactoriza.*

Cuando se está programando una aplicación con TDD, como hemos visto en ediciones anteriores, el proceso de desarrollo se está dividiendo en dos acciones concretas: programar y *refactorizar*. Esto es, a medida que vamos creando nuevos métodos, vamos *refactorizando* el código para eliminar redundancias y en definitiva, hacer el código -del test- más legible y así obtener un mejor rendimiento. Pero no estamos refactorizando el SUT constantemente (aunque sí lo *refactorizamos*), puesto que éste, tiene un momento y lugar para ser *refactorizado*.

---

10 Kent Beck, uno de los creadores de eXtreme Programming, es quien introdujo el término "bad smells" (malos olores) para referirse de manera global, a aquellas expresiones y algoritmos poco claros que generan confusión en el código fuente de un sistema, tornándolo más complejo de lo que debería ser.

11 En el béisbol, un strike es una anotación negativa para el bateador ofensivo, cuando la pelota no es lanzada hacia el diamante. Al tercer strike anotado, termina el turno del bateador.

La *refactorización* del SUT, implica que lo primero que debemos hacer es cumplir el objetivo (programar aquello que se necesita) y luego, *refactorizar* el código del SUT cuando:

- Se agregue un nuevo método;
- Se corrija un *bug*;
- Se haga una revisión de código;

Pero siempre, respetando la regla de "los tres *strikes*". Una vez identificado el momento, solo será cuestión de identificar el problema a fin de poder elegir la solución indicada.

# Una solución a cada problema

Como comentamos anteriormente, no haremos una extensa lista de problemas, sino que nos centraremos en problemas generales. Muchas de las soluciones sugeridas en este artículo, pueden hallarse en SourceMaking.com[12], sitio donde se puede encontrar una completa clasificación de problemas[13] y sus respectivas soluciones[14].

## PROBLEMA: Variables de uso temporal mal implementadas

En principio, definiremos a las variables de uso temporal, como aquellas variables que son asignadas en el ámbito local de un método de clase y son necesarias temporalmente, solo en ese método, sin ser llamadas o requeridas por otros métodos.

> Por favor, notar que cuando se habla de métodos de clase, el significado podría aplicarse también a funciones y procedimientos.

Generalmente representan un problema en tres casos, los cuáles veremos a continuación.

### Caso 1: Variables de uso temporal que definen una acción concreta:

```
# Código PHP
$var = ($a * $b ) / (int)$c;
```

12 http://sourcemaking.com/refactoring. Nótese que algunas de las técnicas expuestas en el sitio Web referido, no se mencionan en este curso, por considerarlas poco apropiadas. Esto es debido a que algunas prácticas son más específicas de lenguajes como Java, mientras que a otras, las considero contrarias a las buenas prácticas de la programación orientada a objetos y por lo tanto, contraproducentes.
13 "Bad Smells in Code" http://sourcemaking.com/refactoring/bad-smells-in-code
14 Diferentes técnicas de refactorización: http://sourcemaking.com/refactoring

```
# Código Python
var = (a * b) / int(c)
```

En el ejemplo anterior, vemos una variable de uso temporal, que define una acción concreta: dividir el producto de dos factores. Esto representa un problema, ya que las acciones son responsabilidad de los métodos y no de las variables. En estos casos, la solución, es transferir la responsabilidad de la acción a un método:

```
# Código PHP
$var = dividir($a, $b, $c);

function dividir($a, $b, $c) {
    return ($a * $b ) / (int)$c;
}


# Código Python
var = dividir(a, b, c)

def dividir(a, b, c):
    return (a * b) / int(c)
```

Las variables de uso temporal que definen un valor directo: $var = 15; o por el retorno de la llamada a una función: $var = strlen($variable); no necesitan transferir su responsabilidad a otro método.

**Caso 2. Variables de uso temporal son requeridas por más de un método:**

```
# Código PHP
function metodo_a() {
    $a = 15;
    $b = 100;
    $c = 2;
    $var = self::dividir($a, $b, $c);
    // continuar...
}

private static function dividir($a, $b, $c) {
    return ($a * $b ) / $c;
}


# Código Python
def metodo_a(self):
    a = 15
    b = 100
    c = 2
    var = self._dividir(a, b, c)
    # continua...
```

```
def _dividir(self, a, b, c):
    return (a * b) / c
```

En el ejemplo, anterior, las variables temporales `$a`, `$b` y `$c`, son requeridas por dos métodos y se están definiendo como tales en un método, necesitando ser pasadas como parámetros. Aquí, la solución, será convertir las variables temporales, en propiedades de clase:

```
# Código PHP
function metodo_a() {
    self::$a = 15;
    self::$b = 100;
    self::$c = 2;
    $var = self::dividir();
    // continuar...
}

private static function dividir() {
    return (self::$a * self::$b ) / self::$c;
}


# Código Python
def metodo_a(self):
    self.a = 15
    self.b = 100
    self.c = 2
    var = self._dividir()
    # continua...

def _dividir(self):
    return (self.a * self.b) / self.c
```

**Caso 3. Variables de uso temporal que reasignan parámetros:**

```
# Código PHP
function foo($a) {
    $a = strtoupper($a);
    // continuar ...
}

# Código Python
def foo(a):
    a = a.upper()
    # continua...
```

En casos como éste, la confusión puede ser grande: un parámetro es un parámetro y una variable temporal, una variable temporal. Es entonces, cuando variables temporales no deben tener el mismo nombre que los parámetros:

```
# Código PHP
function foo($a) {
    $b = strtoupper($a);
    // continuar ...
}


# Código Python
def foo(a):
    b = a.upper()
    # continua...
```

## PROBLEMA: Métodos que reciben parámetros

Aquí debe hacerse una notable distinción entre parámetros, variables de uso temporal y propiedades de clase. Y esta distinción, está dada por la finalidad de cada una:

- Las variables de uso temporal, como hemos visto antes, están destinadas a definir un valor concreto al cual se hará referencia solo en el ámbito donde se haya definido.
- Las propiedades de clase, son características inherentes al objeto a las cuales se hará referencia desde diversos ámbitos.
- Y finalmente, los parámetros, serán valores adicionales, que no pueden ser considerados propiedades del objeto pero que sin embargo, son requeridos para que una acción, por ejemplo, modifique las propiedades de un objeto.

Veamos algunos casos de mal uso de parámetros:

```
/*
    Parámetros cuya única finalidad es modificar de forma directa
    el valor de una o más propiedades
*/
class Producto {

    function __construct($nombre, $precio, $barcode) {
        $this->nombre = $nombre;
        $this->precio = $precio;
        $this->barcode = $barcode;
    }

}

$nombre = 'Agua mineral';
$precio = 5.75;
$barcode = 6543922234321;
$producto = new Producto($nombre, $precio, $barcode);

/*
    Métodos que solo recurren a sus parámetros actuando como funciones
*/
class Usuario(object):
```

```
    def validar_usuario(self, username, passwd):
        if username == 'pepe' and passwd == '123':
            return True
```

Como regla general, los parámetros deben ser evitados toda vez que sea posible, reemplazándolos por propiedades de clase. A no ser que una propiedad deba componerse de un objeto, los valores de la misma deberán ser modificados de forma directa evitando para ello el uso de parámetros:

```
/*
    Parámetros cuya única finalidad es modificar de forma directa
    el valor de una o más propiedades
*/
class Producto {

    function __construct() {
        $this->nombre = '';
        $this->precio = '';
        $this->barcode = '';
    }

}

$producto = new Producto();
$producto->nombre = 'Agua mineral';
$producto->precio = 5.75;
$producto->barcode = 6543922234321;


/*
    Métodos que solo recurren a sus parámetros actuando como funciones
*/
class Usuario(object):

    def validar_usuario(self):
        if self.username == 'pepe' and self.passwd == '123':
            return True
```

## PROBLEMA: Expresiones extensas

Muchas veces, podremos encontrarnos con expresiones que debido a su extensión, se hacen difíciles de leer e inducen a una gran confusión:

```
# Código PHP
return ((in_array('abc', $array) || in_array('bcd', $array)) && (in_array('cde',
$array) || in_array('def', $array))) ? 'OK' : 'ERROR';

# Código Python
return 'OK' if (('abc' in lista or 'bcd' in lista) and ('cde' in lista or 'def' in
lista)) else 'ERROR'
```

Cuando estamos en presencia de expresiones tan extensas, lo mejor es -aquí sí- utilizar variables de uso temporal para simplificar dichas expresiones:

```
# Código PHP
$a = in_array('abc', $array);
$b = in_array('bcd', $array);
$c = in_array('cde', $array);
$d = in_array('def', $array);
$ab = ($a || $b);
$cd = ($c || $d);

return ($ab && $cd) ? 'OK' : 'ERROR';

# Código Python
a = 'abc' in lista
b = 'bcd' in lista
c = 'cde' in lista
d = 'def' in lista
ab = a or b
cd = c or d

return 'OK' if ab and cd else 'ERROR'
```

# Configurando GitWeb en Ubuntu Server

**Gitweb es una GUI que te permite poder gestionar todos los repositorios de git en tu servidor y la ventaja que yo le encuentro es sobre todo para *monitorear* los avances de los proyectos que gestionamos desde cualquier lugar.**

*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**

Gitweb es un script en CGI, que nos permite gestionar visualmente los cambios en nuestros proyectos pudiendo ver contenido de los registros, ramas especificas, diferencias aplicadas en los cambios, entre otras. También te permite descargar la versión de un archivo o la versión de un proyecto.

Por supuesto existen muchas GUI en el mercado -sobre todo, versiones de escritorio-; son muy buenas y ofrecen mayor variedad para el desarrollador, pero sin embargo, aprecio *gitweb* pues puedo *monitorear* desde mi celular, que los cambios que se debieron subir, ya están ejecutados.

Ahora si manos a la obra:

```
# Instalación y primeros pasos
root@dev:~# apt-get install gitweb
root@dev:~# cd /var/www
En esta ruta tenemos que crear una carpeta gitweb
root@dev:/var/www# mkdir gitweb
```

Los archivos de gitweb suelen estar en `/usr/share/gitweb` y los necesitamos en el directorio `/var/www/gitweb`. Para ello, crearemos un vínculo simbólico

```
root@dev:/var/www# cd gitweb
root@dev:/var/www/gitweb# ln -s /usr/share/gitweb/* .
root@dev:/var/www# ls -l
```

Ahora podemos ver como los archivos están *linkeados* a los otros:

```
root@dev:/var/www/gitweb# ls -l
total 0
lrwxrwxrwx 1 root root 28 2013-04-12 21:55 gitweb.cgi -> /usr/share/gitweb/gitwe
b.cgi
lrwxrwxrwx 1 root root 27 2013-04-12 21:55 index.cgi -> /usr/share/gitweb/index.
cgi
lrwxrwxrwx 1 root root 24 2013-04-12 21:55 static -> /usr/share/gitweb/static
root@dev:/var/www/gitweb#
```

Ahora tenemos que hacer que gitweb apunte a nuestros repositorios que están localizados en `/home/git/dev`

```
root@dev:~# nano /etc/gitweb.conf
Remplazar $projectroot = "/var/cache/git"; por $projectroot = "/home/git/dev";
```

Creamos un Virtual Host:

```
root@dev:~# nano /etc/apache2/conf.d/gitweb
```

Donde se debe copiar esto:

```
RewriteEngine on
RewriteRule ^[a-zA-Z0-9_\-]+\.git/?(\?.*)?$ /gitweb.cgi%{REQUEST_URI} [L,PT]

Alias /gitweb /usr/share/gitweb

<Directory /usr/share/gitweb>
   Options Indexes FollowSymlinks ExecCGI
   DirectoryIndex /cgi-bin/gitweb.cgi
   AllowOverride None
</Directory>
```

Y reiniciamos Apache:

```
root@dev:~# /etc/init.d/apache2 restart
```

Si aparece el siguiente error:

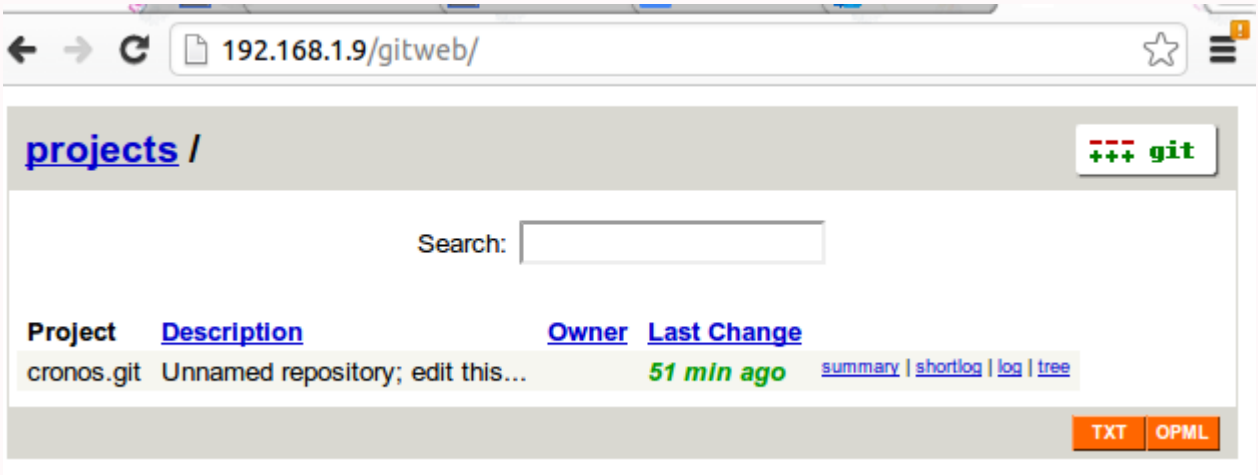
```
Syntax error on line 1 of /etc/apache2/conf.d/gitweb:
Invalid command 'RewriteEngine', perhaps misspelled or defined by a module not i
ncluded in the server configuration
Action 'configtest' failed.
The Apache error log may have more information.
   ...fail!
```

Habilitamos el módulo `rewrite` de Apache:

```
root@dev:~# a2enmod rewrite
```

```
Enabling module rewrite.
Run '/etc/init.d/apache2 restart' to activate new configuration!
```

Ahora entremos a http://192.168.1.9/gitweb

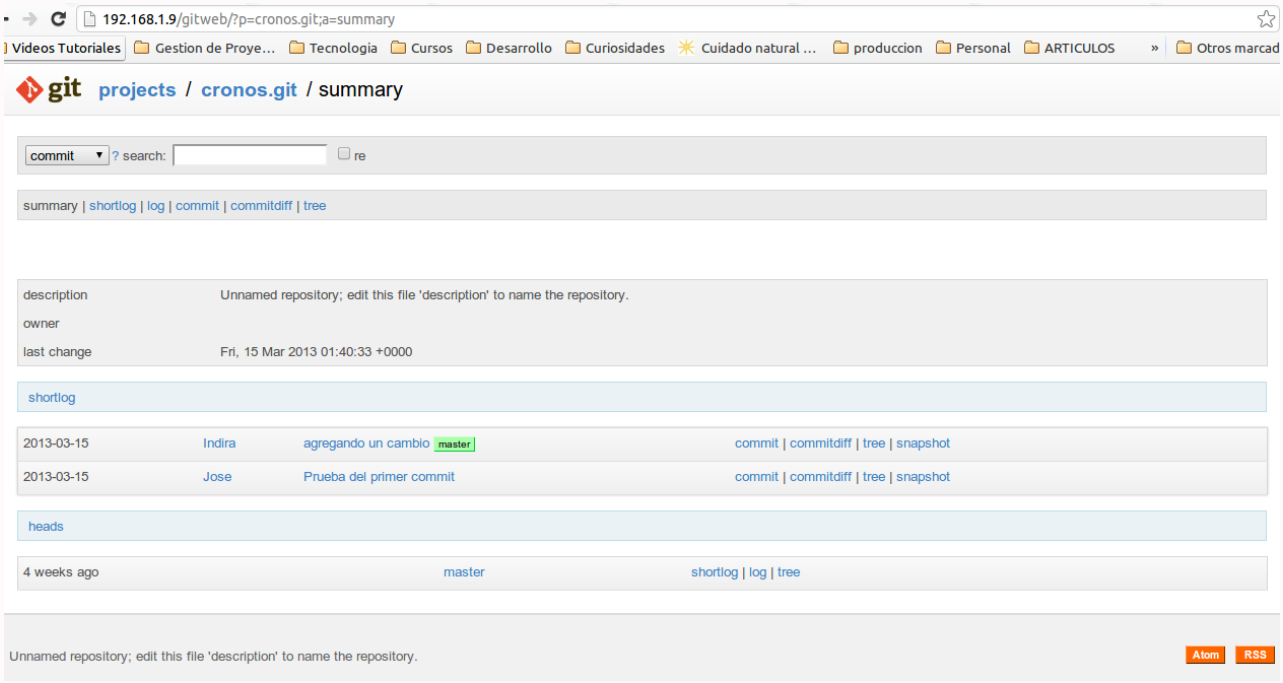



Como verán ya instalamos el *gitweb* y podemos ver nuestro repositorio.

La interfaz no es nada amigable, pero podemos recurrir al tema de *gitweb* alojado en http://kogakure.github.io/gitweb-theme/

```
root@dev:/usr/share/gitweb/static# git clone git://github.com/kogakure/gitweb-theme
root@dev:/usr/share/gitweb/static# cd ..
root@dev:/usr/share/gitweb# cp -R static/ static_original/
root@dev:/usr/share/gitweb# rm -R static/*
root@dev:/usr/share/gitweb/static# ln -s gitweb-theme/gitweb.css gitweb.css
root@dev:/usr/share/gitweb/static# ln -s gitweb-theme/gitweb.js gitweb.js
root@dev:/usr/share/gitweb/static# ln -s gitweb-theme/git-logo.png git-logo.png
root@dev:/usr/share/gitweb/static# ln -s gitweb-theme/git-favicon.png
```

Este seria el resultado:





# Draw.io: diagramas para tus proyectos

**A todo desarrollador siempre le va a ser de gran utilidad contar con herramientas que le permitan dibujar diagramas (de todo tipo) de acuerdo a lo que necesite. Draw.io es una gran herramienta que ayudará a que esta tarea no se torne tediosa ni aburrida, sino por el contrario, fácil y simple.**

*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**

Todos los desarrolladores necesitan de una herramienta que les permita dibujar diagramas para distintos casos según sea la necesidad, como por ejemplo, cuando se trata de diagramas UML, diagramas de flujo, diagramas BPMN, *mockups*, entre muchos más. Pero el hecho de elegir la herramienta indicada también se vuelve una preocupación y es por esto que contar con una alternativa *online* de uso fácil y sencillo es genial y éste es el caso de Draw.io.

Draw.io es una aplicación para realizar diagramas en línea que funciona en todo lugar y está construida para la velocidad, confiabilidad y simplicidad. Usa la librería de JavaScript `mxGraph`; cuenta con una amplia gama de configuración visual al igual que una aplicación web con una gran variedad de opciones como una larga colección de iconos, colaboración en tiempo real y posibilidad de compartir *widgets* embebidos.

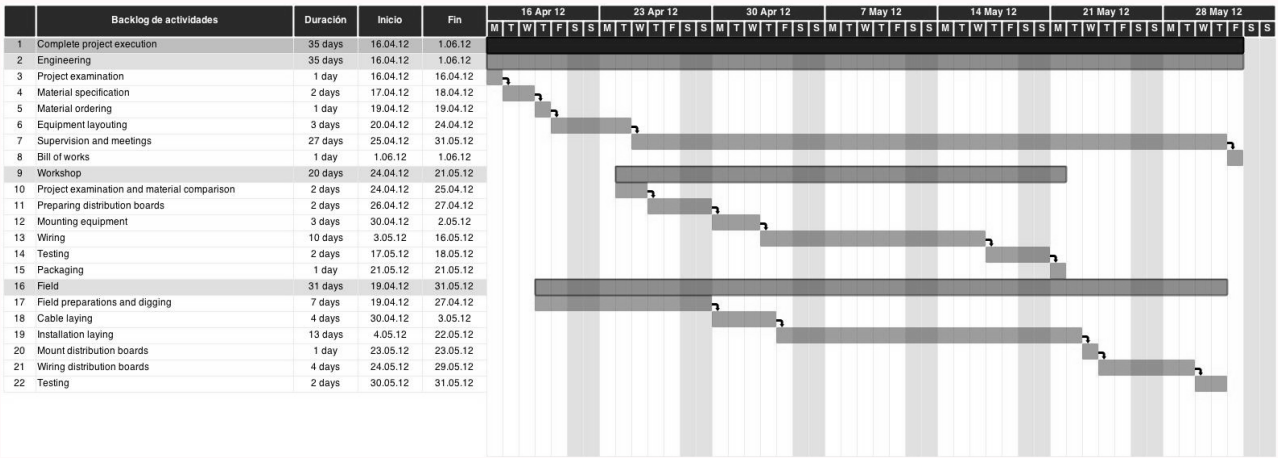
Respecto a sus integraciones múltiples, draw.io opcionalmente se integra por completo con Google™ Drive utilizando el almacenamiento en la nube de la cuenta de Google™. Draw.io es libre de licenciar cualquier número de usuarios en cualquier entorno y se beneficia de un portal de apoyo comunitario.

# Características

Draw.io puede tener usuarios ilimitados como también dibujos ilimitados; cuenta con una amplia gama de librerías de formas; puede exportar a PNG/JPG/XML/SVG; tiene integración con Google™ Drive y colaboración en tiempo real; puede ser usado en cualquier navegador con HTML 5, contando con soporte para navegadores privativos; cuenta además, con la posibilidad de embeber *widgets* y sobretodo es de uso gratuito.

Al ingresar a draw.io[15] encontraremos una interfaz bastante simple para empezar a dibujar. Podemos comenzar probando la aplicación para realizar un *mockup* básico:

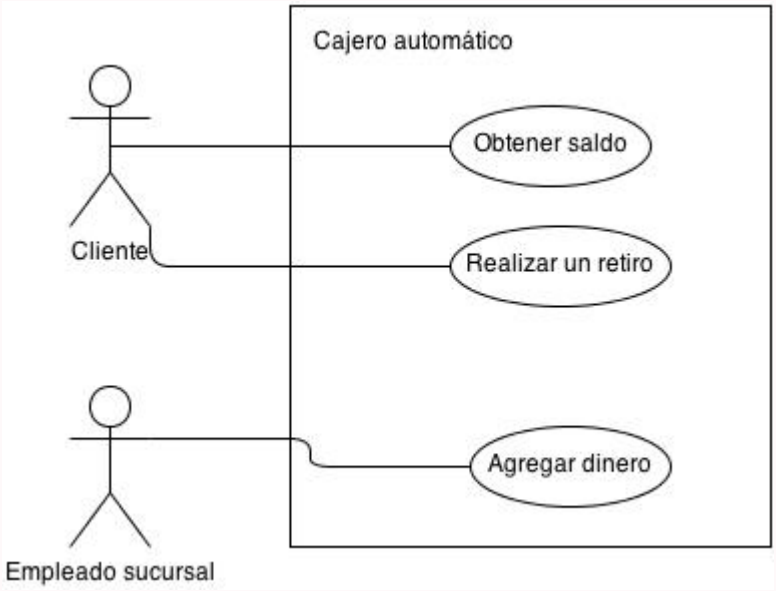

También cuenta con plantillas que pueden modificarse según lo que se desea:



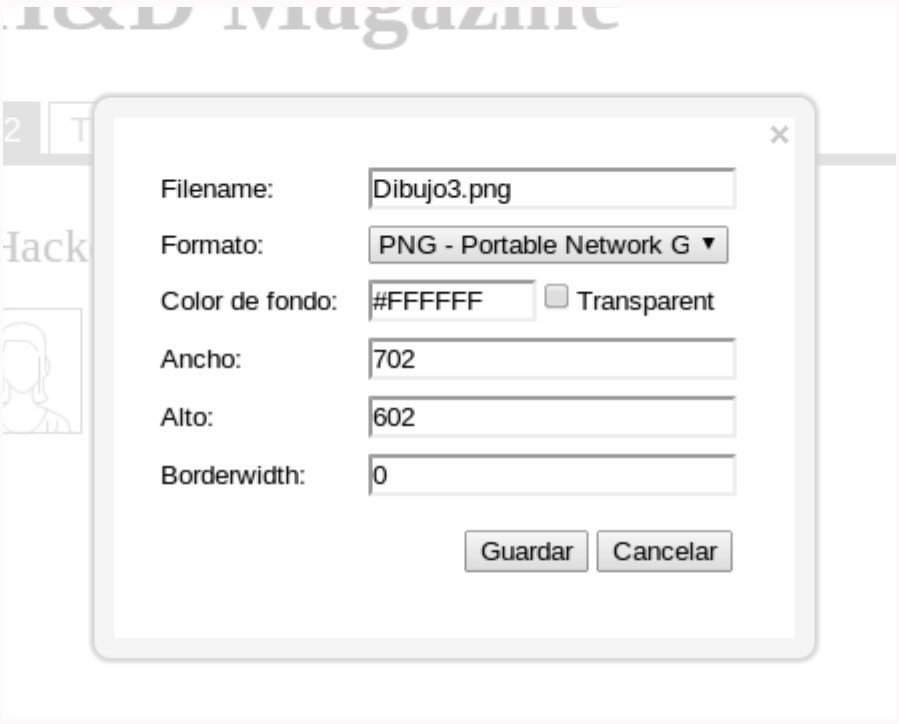
| | Backlog de actividades | Duración | Inicio | Fin |
|---|---|---|---|---|
| 1 | Complete project execution | 35 days | 16.04.12 | 1.06.12 |
| 2 | Engineering | 35 days | 16.04.12 | 1.06.12 |
| 3 | Project examination | 1 day | 16.04.12 | 16.04.12 |
| 4 | Material specification | 2 days | 17.04.12 | 18.04.12 |
| 5 | Material ordering | 1 day | 19.04.12 | 19.04.12 |
| 6 | Equipment layouting | 3 days | 20.04.12 | 24.04.12 |
| 7 | Supervision and meetings | 27 days | 25.04.12 | 31.05.12 |
| 8 | Bill of works | 1 day | 1.06.12 | 1.06.12 |
| 9 | Workshop | 20 days | 24.04.12 | 21.05.12 |
| 10 | Project examination and material comparison | 2 days | 24.04.12 | 25.04.12 |
| 11 | Preparing distribution boards | 2 days | 26.04.12 | 27.04.12 |
| 12 | Mounting equipment | 3 days | 30.04.12 | 2.05.12 |
| 13 | Wiring | 10 days | 3.05.12 | 16.05.12 |
| 14 | Testing | 2 days | 17.05.12 | 18.05.12 |
| 15 | Packaging | 1 day | 21.05.12 | 21.05.12 |
| 16 | Field | 31 days | 19.04.12 | 31.05.12 |
| 17 | Field preparations and digging | 7 days | 19.04.12 | 27.04.12 |
| 18 | Cable laying | 4 days | 30.04.12 | 3.05.12 |
| 19 | Installation laying | 13 days | 4.05.12 | 22.05.12 |
| 20 | Mount distribution boards | 1 day | 23.05.12 | 23.05.12 |
| 21 | Wiring distribution boards | 4 days | 24.05.12 | 29.05.12 |
| 22 | Testing | 2 days | 30.05.12 | 31.05.12 |

15 https://www.draw.io/

También podríamos realizar un diagrama UML:




Después de tener listo el dibujo guardamos el archivo con `.xml`, también podemos exportarlo en distintos formatos y luego poder hacer uso de ellos en todo lugar que deseemos e incluso embeberlo, ya que al presionar en la opción nos da el código para poder lograrlo.




El modelo de negocio de draw.io es darse a conocer mediante esta herramienta ya que ellos creen firmemente en que si no confían en el modelo no lo utilizarían nunca, entonces es la manera de hacerle marketing, permitiendo que los usuarios puedan tener interacción con este.

Si estás interesado en colaborar con el proyecto o quizás tener conversaciones con gente involucrada, puedes entrar a los foros[16] y ponerte en comunicación con ellos e incluso por las redes sociales que aparecen en la página principal de draw.io.

> *Draw.io es una aplicación genial que te permite realizar los diagramas que necesites en el proceso de desarrollo de software y entre sus ventajas principales está la facilidad de poder realizar diagramas de manera sencilla y rápida.*

16 https://jgraph.freshdesk.com/support/discussions

## GNU/Linux para programadores:

# Automatizando tus aplicaciones con Cron

**A la hora de resolver determinados requerimientos de Software es muy posible que la solución ideal implique la ejecución de ciertas tareas de forma automática y esto, solo será posible gracias *Cron*.**

*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**

Cron es el administrador de procesos en segundo plano de los sistemas operativos GNU/Linux, permitiendo definir en un archivo `crontab`, la ejecución de comandos y/o *scripts* a intervalos regulares de tiempo.

Para **visualizar las tareas existentes**, basta con ejecutar `crontab` con el parámetro `-l`, como se muestra a continuación:

```
user@host:~$ crontab -l
```

El archivo `crontab` posee una sintaxis muy estricta en la cual, cada tarea debe especificarse en una línea. A la vez, cada línea deberá contar con la siguiente estructura:

```
# Para programar la ejecución de un script
periodicidad /ruta/al/script

# Para programar la ejecución de un comando
periodicidad comando
```

La periodicidad de ejecución puede indicarse con alguno de los intervalos preestablecidos:

```
@daily /home/eugenia/projects/myapp/bin/backup_db.sh
```

Entre los intervalos preestablecidos, podemos encontrar:

```
@yearly          Se ejecuta una vez al año, a las 0 hs del 1 de enero
                 Alias: @annually

@monthly         Se ejecuta una vez al mes, a las 0 hs del primer día de cada mes

@weekly          Se ejecuta una vez a la semana, a las 0 hs del primer día de la
                 semana (domingo)

@daily           Se ejecuta una vez al día, a las 0 hs
                 Alias: @midnight

@hourly          Se ejecuta una vez por hora
```

Sin embargo, la flexibilidad de *cron* es muy amplia y nos permite ser más específicos al indicar un intervalo, utilizando la sintaxis:

```
MINUTO HORA DIA-DEL-MES MES DIA-DE-LA-SEMANA tarea

Donde cada valor será reemplazo por:

          MINUTO:  Un entero entre 0 y 59
            HORA:  Un entero entre 0 y 23
     DIA-DEL-MES:  Un entero entre 1 y 31
             MES:  Un entero entre 1 y 12
DIA-DE-LA-SEMANA:  Un entero entre 0 y 7 donde 0 y 7 representan al día domingo
```

Cuando no se desea indicar un valor específico, se deberá utilizar el asterisco:

```
0 0 5 3 * tar -czf /srv/backup/eugenia.tar.gz /home/eugenia

Lo anterior puede entenderse como:
Minuto:            0
Hora:              0
Día del mes:       5
Mes:               3 (marzo)
Día de la semana:  * (sin especificar)
Tarea:             tar -czf /srv/backup/eugenia.tar.gz /home/eugenia
```

Es decir que todos los 5 de marzo a las 0 horas, se hará una copia de respaldo de la *home* de mi usuario.

Claro que en vez de la ejecución de un comando, podría tratarse de la ejecución de un *script*. En ese caso, el ***script*** debe cuidar de **indicar en la primera línea, el binario con el cual debe ejecutarse**. Por ejemplo:

```
BASH:
#!/bin/bash

PERL:
#!/usr/bin/perl

PHP:
#!/usr/bin/php

PYTHON:
#!/usr/bin/python
```

Y también, debe tener **permisos de ejecución**:

```
chmod +x /ruta/al/script
```

Para **agregar tareas** al archivo `crontab`, se debe ejecutar el siguiente comando:

```
crontab -u usuario -e

Por ejemplo, para agregar tareas al crontab del usuario juan:
root@host:~# crontab -u juan -e
```

El parámetro `-u` será opcional si lo que se desea es agregar una tarea al `crontab` del usuario que está ejecutando dicho comando:

```
eugenia@hots:~$ crontab -e
Agregará una tarea al crontab del usuario eugenia
```

Si no tienes un editor de texto por defecto, la primera vez que ejecutes `crontab -e` te pedirá que elijas el editor predeterminado.

Pero aquí, no termina todo. `Cron`, también permite especificar intervalos mucho más complejos y específicos y para ello, dispone de los siguientes operadores:

```
/   Salta la unidad especificada
```

```
0 */6 * * * rm -r /srv/www/myapp/temp/*
```
*Elimina todos los archivos del directorio /srv/www/myapp/temp/* ***cada 6 horas***

```
*/10 * * * * rm -r /srv/www/myapp/temp/*
```
*Elimina todos los archivos del mismo directorio pero lo hace* ***cada 10 minutos***

**,  La coma, permite especificar una lista precisa**

```
0 0 1,5,15 * * rm -r /srv/www/myapp/temp/*
```
*Elimina todos los archivos del mismo directorio los días 1, 5 y 15 de cada mes a las 0 horas*

```
0 2 * 3,9 * rm -r /srv/www/myapp/temp/*
```
*Elimina todos los archivos del mismo directorio, todos los días a las 2:00 a.m. En los meses de marzo y septiembre*

**-  El guión medio, permite especificar un rango**

```
45 6 10-15 * * rm -r /srv/www/myapp/temp/*
```
*Elimina todos los archivos del mismo directorio entre los días 1 y 5 de cada mes a las 6:45 a.m.*

```
0 0 * 9-12 * rm -r /srv/www/myapp/temp/*
```
*Elimina todos los archivos del mismo directorio, todos los días a las 0 horas entre los meses de septiembre y diciembre*

Cron no permite ejecutar de forma directa, aplicaciones a través de entorno gráfico. Por ejemplo, si todos los días a las 17:00 hs. quisieras ejecutar el navegador Firefox, la siguiente instrucción, **NO funcionará**:

```
0 17 * * * firefox  # No funcionará
```

**Para que funcione**, se debe agregar la instrucción **DISPLAY=":0"** precediendo al comando, como se muestra a continuación:

```
0 17 * * * DISPLAY=":0" firefox
```

# Pharoah (Faraón)

por **joan stark (jgs)**

www.ascii-art.com

# U! Tu zona exclusiva

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

## Carta de un Lector

**Hola !, muy buen día :) mi nombre es Michel, soy un estudiante de ingeniería en sistemas de México.**

primero que nada, me gustaría felicitar a todo su gran equipo de hackers y programadores, ya que son gente muy talentosa (...)

me decidí a escribirles ya que me quedé muy picado con la revista número 5 que sacaron, especialmente en la parte de Milagros Infante, yo soy nuevo con python, recién comienzo a programar y me gustó python como primer lenguaje.

he comenzado con todo lo básico, desde operaciones con if, for, while, etc., y la verdad que me ha encantado python, pero no soy muy bueno encontrando buen material para aprender a programar mejor, sobre todo manuales en español, sinceramente agradecería muchísimo que pudieran recomendarme manuales, videos o cualquier material para aprender acerca de python.

de ante mano muchas gracias por tomarse la molestia de leer este correo, espero una respuesta :)

y de nuevo felicitarlas por el gran trabajo que han llevado acabo con su revista :)

### Respuesta de Milagros Infante Montero:

Hola Michel: Saludos desde Perù, es genial que quieras entrar en el mundo dr Python porque creeme que no te arrepentiras. Como te das cuenta en internet hay miles de páginas con muchísima información, pero para que empieces de la mejor manera, aprendiendo correctamente y sobretodo teniendole mucho gusto, te recomiendo los siguientes enlaces: [0] Aquí puedes encontrar muchos buenos libros de Python, [1] este es el tutorial escrito por Eugenia Bahit en Maestros del Web, [2] aquí tienes un libro escrito también por Eugenia y aquí [3] un tutorial también bueno de MundoGeek. Espero todos te sean de mucha utilidad, estoy segura de que si, sobretodo serán la mejor manera para que empieces en este maravilloso mundo y que además tengas los mejores happy codings con Python. <o/

[0] http://python.org.ar/pyar/AprendiendoPython
[1] http://www.maestrosdelweb.com/editorial/guia-python/
[2] http://cursos.eugeniabahit.com/static/pdf/material-sin-personalizar-python.pdf
[3] http://mundogeek.net/tutorial-python/

## Mensajes de Lectores

**Bryan Rodríguez**
Me gustó su revista. Tiene muchos temas interesantes tanto para novatos como para usuarios novatos. Voy a escribir un artículo sobre ustedes en mi blog (http://ordenadores.eninternet.es).

***Respuesta de H&D:*** *¡Gracias Bryan!*

**Juan Pablo Martínez Delbujio**
Bueno, soy admirador de ustedes, me descargo todas sus ediciones y me encantan! Y les queria dar la idea, de al igual que crearon la capa de abstraccion segura con mysqli pudieran crear una con mongodb, para (obviamente) conectarse a esta base de datos. Lo que yo me refiero seria, crear una forma segura de conectarse usando esto:
http://www.php.net/manual/es/book.mongo.php

***Respuesta de H&D:*** *Gracias por tu mensaje, Juan Pablo! Y tienes nuestra promesa de que publicaremos algo al respecto :)*



¡GRACIAS POR LEERNOS!