AÑO ---------- 0
NÚMERO -------- 5
FECHA: 2013-03-25

#5

"SWEET"

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
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 #4

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 Permanente

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

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

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

Sergio Infante Montero
Ingeniero de Software
neosergio.net

Indira Burga
Ingeniera de Sistemas
about.me/indirabm

Colaboran:

Celia Cintas, Laura Mora

Difunden:

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

DESARROLLO WEB

LESS – El lenguaje dinámico de estilos

Una manera de extender las hojas de estilo y convertir al CSS en una especie de lenguaje de programación.

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**

Uno de los problemas más comunes cuando se está construyendo, ampliando o modificando una hoja de estilo, es el no poder usar variables dentro del CSS. Incluso podemos echar de menos el uso de funciones, parámetros y tratar al CSS como una especie de lenguaje de programación.

Es así cómo Alexis Sellier (@cloudhead) influenciado por el trabajo de Hampton Catlin (@hcatlin) con el SASS[1], un metalenguaje de CSS, en 2009 diseña **LESS**[2] como un lenguaje de estilos dinámico.

LESS es un proyecto open source, que fue escrito al principio en Ruby, pero que luego fue pasado a Javascript.

Lo resaltante de LESS es que puede funcionar desde el lado del cliente, el lado del servidor y también puede ser compilado para generar CSS, así como también permite el uso de mecanismos como variables, anidamiento, mixins, funciones y operaciones.

La manera de usarlo

La manera más simple de empezar a desarrollar con LESS es del lado del cliente. Sin embargo, para entornos de producción y especialmente si se necesita mejorar el

1 http://sass-lang.com/
2 http://lesscss.org/

desempeño es muy importante pre-compilar less, usando node.js[3] o alguna herramienta de terceros[4].

> Puedes usar http://less2css.org/ para poder probar el preprocesamiento. Es una aplicación on-line y puedes ver el resultado en el instante.

En este artículo veremos del lado del cliente y la configuración por defecto. Para ello debemos descargar e incluir `less.js` dentro de nuestra página:

```
<script src="less.js" type="text/javascript"></script>
```

Cada hoja de estilo que hagamos con LESS, la debemos poner de la siguiente manera:

```
<link rel="stylesheet/less" type="text/css" href="estilos.less" />
```

Las variables

Son una de las características que mayor utilidad podemos encontrar. Veámoslo con un ejemplo. Tenemos los siguientes estilos:

```
#elemento1 {
    color: #ff0000;
}
#elemento2 {
    color: #ff0000; float: left;
}
#elemento3 {
    font-size: 2em; color:#ff0000; display:inline;
}
```

Imaginemos que queremos modificar el color (que es el mismo para todos los elementos). Escribiendo como el CSS tradicional, el cambio tiene que hacerse línea por línea y basta con imaginar un archivo más grande para darnos cuenta de lo terrible y confuso que puede tornarse un cambio de color en una hoja de estilos.

Para este ejemplo entonces, con LESS tendríamos lo siguiente:

```
@color: #ff0000;
#elemento1 {
    color: @color;
}
```

3 https://github.com/cloudhead/less.js/wiki/Command-Line-use-of-LESS
4 https://github.com/cloudhead/less.js/wiki/GUI-compilers-that-use-LESS.js

```
#elemento2 {
    color: @color; float: left;
}
#elemento3 {
    font-size: 2em; color: @color; display: inline;
}
```


De esta manera si deseamos cambiar el color sólo tendríamos que cambiarlo una vez, sin importar la cantidad de estilos dentro de la hoja.

Las variables pueden ser utilizadas para definir colores, tamaños, porcentajes, valores y demás.

El anidamiento

LESS también permite hacer anidamiento de estilos o lo que se le conoce como estructura en cascada. Veamos un ejemplo:

```
#principal { margin: 10px; }
#principal .menu { float: left; font-size: 12px; }
#principal .dialogo { width: 400px; }
#principal .boton { margin:0 auto; }
#principal .boton:hover { background-color:#660066; }
```

En la sintaxis LESS sería de la siguiente manera:

```
#principal {
    margin: 10px;
    .menu { float: left; font-size: 12px; }
    .dialogo { width: 400px; }
    .boton {
          margin: 0 auto;
          &:hover { background-color: #660066; }
    }
}
```

O si se desea, también de esta manera que luce más ordenada y se entiende mejor:

```
#principal       { margin: 10px;
    .menu        { float: left; font-size: 12px; }
    .dialogo     { width: 400px; }
    .boton       { margin: 0 auto;
      &:hover    { background-color: #660066; }
    }
```

```
}
```

Este tipo de codificación ayuda incluso a entender la estructura DOM de la página; también es muy útil para hacer anidamientos de *media queries*, ayudando así a que sean muy entendibles los estilos que se están escribiendo.

Los Mixin

Un *mixin* es una clase que permite ser heredada por una subclase, pero no está pensada para ser autónoma. Existen varios lenguajes de programación que usan *mixins* como por ejemplo: C#, Dart, Javascript, Perl, Python, PHP, Ruby, Scala, Vala, entre otros.

En LESS podemos tener el siguiente ejemplo:

```
.resaltado {
    font-size: 1.5em;
    color: #ff9900;
}
#parrafo {
    margin: 5px;
    .resaltado;
}
footer {
    padding: 10px;
    .resaltado;
}
```

Esto provocaría el siguiente resultado en CSS:

```
#parrafo {
    margin: 5px;
    font-size: 1.5em;
    color: #ff9900;
}

footer {
    padding: 10px;
    font-size: 1.5em;
    color:#ff9900;
}
```

Se puede mezclar obviamente con variables y tendríamos un mejor resultado. Los *mixins* pueden aceptar parámetros, por ejemplo:

```
.margenes (@espacio: 3px) {
    margin-top: @espacio;
    margin-bottom: @espacio;
}
```

```
nav {
    .margenes;
}
section {
    .margenes(5px);
}
footer {
    .margenes(10px);
}
```

Esto dará como resultado:

```
nav {
    margin-top: 3px; margin-bottom: 3px;
}
section {
    margin-top: 5px; margin-bottom: 5px;
}
footer {
    margin-top: 10px; margin-bottom: 10px;
}
```

Se pueden aplicar otras combinaciones, incluso utilizar las variables `@arguments`, `@rest` entre otras para hacer *mixins* avanzado. Todo esto está disponible en la documentación del proyecto[5].

Las funciones

LESS tiene varias funciones que permiten manipular cadenas, hacer cálculos aritméticos y transformar colores. Todas estas funciones están documentadas en las referencias de funciones en el proyecto[6].

Aquí un breve ejemplo de lo que puede hacerse con las funciones:

```
@color: #ff0000;
.alerta {
    color: saturate(@color, 10%);
    background-color: average(@color, #00ff00);
}
```

Esto daría como resultado:

```
.alerta {
  color: #ff0000;
```

5 http://lesscss.org/#docs
6 http://lesscss.org/#reference

```
  background-color: #808000;
}
```

Las operaciones

Cualquier número, variable o color es susceptible a ser operada. Acá unos ejemplos de como se hacen:

```
@relleno: (10px + 20);
header {
    padding: @relleno * 2;
}
```

Esto dará como resultado:

```
header {
  padding: 60px;
}
```

O con colores:

```
@color: #ffff00;
body {
    background-color:(@color / 2);
}
```

Lo cual dará como resultado:

```
body {
  background-color: #808000;
}
```

Estas son sólo algunas cosas que se pueden hacer con LESS. Si se desea contribuir con el proyecto, el código fuente esta alojado en github[7]. Y no olvidar revisar la documentación y practicar.

La práctica convierte al alumno en maestro

7 https://github.com/cloudhead/less.js

Estructura de datos en Python

Al programar es muy importante contar con una forma de organizar un conjunto de datos para facilitar así su manipulación, con esto me refiero a contar con una estructura de datos que nos permita definir la organización e interrelación de datos y operaciones, la cual la veremos enfocada a Python.

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**

Cuando hablamos de programación, la estructura de datos es una forma de organizar el *dataset* (toda información en el sistema) que nos dará la posibilidad de facilitar el manejo del mismo.

Definiremos la organización y relación entre el conjunto de datos y las operaciones que se pueden realizar: añadir un valor, borrar un valor, encontrar un determinado valor (y realizar una operación con él), ordenar valores, aparear estructuras para obtener una nueva, entre muchas otras operaciones. Cada estructura diferirá de otras por la simplicidad o eficiencia en las operaciones.

Conjuntos

Son contenedores de valores sin elementos duplicados y no necesariamente con un orden que puede verse como un *array* (una cadena asociativa). Sus usos básicos son para comprobar la pertenencia y la existencia de elementos repetidos. En Python, los objetos conjunto tienen operaciones matemáticas como unión, intersección, diferencia y diferencia simétrica.

```
>>> armario = ['polo', 'jean', 'falda', 'jean', 'falda']
# crear conjunto sin elementos duplicados
>>> ropa = set(armario)
>>> ropa
set(['polo', 'jean', 'falda'])

# comprobar pertenencia
>>> 'polo' in ropa
True
>>> 'blusa' in ropa
False

# Demostración de las operaciones de conjuntos sobre las letras de dos palabras

>>> a = set('hackdevmagazine')
>>> b = set('hackerteam')
>>> a            #letras sin repetición de a
set(['a', 'c', 'e', 'd', 'g', 'i', 'h', 'k', 'm', 'n', 'v', 'z'])
>>> a – b         # letras de a que no están en b
set(['d', 'g', 'i', 'n', 'v', 'z'])
>>> a & b         # letras que están en a y en b
set(['a', 'c', 'e', 'h', 'k', 'm'])
>>> a | b         # letras que están en a o b
set(['a', 'c', 'e', 'd', 'g', 'i', 'h', 'k', 'm', 'n', 'r', 't', 'v', 'z'])
>>> a ^ b         # letras que están en a y b pero no en los dos
set(['d', 'g', 'i', 'n', 'r', 't', 'v', 'z'])
```

Matriz

Un arreglo o *array* (llamado vector en una dimensión, matriz en dos), son zonas de almacenamiento continuo y ordenado conteniendo elementos del mismo tipo.

```
>>> from array import *
>>> a = array('i',[1,2,3,4,5,6])
>>> a[0]
1
>>> a[4]
5
>>> a[3]+a[1]
6
>>>
```

Lista

Esta es una estructura de datos fundamental. Es una secuencia de nodos en los que se guardan campos de datos arbitrarios y referencias o enlaces al nodo anterior o posterior. Los tipos de listas son:

Lista simple enlazada:

Es la más básica ya que posee un enlace por nodo, el cual apunta al siguiente nodo en la lista y a NULL si es el último nodo.

Lista doblemente enlazada:

En este tipo de lista cada nodo tiene dos enlaces: uno al anterior y otro al NULL si es el primer nodo y otro que apunta al siguiente nodo o a NULL si es el último nodo.

Lista enlazada circular:

El primer y último nodo están unidos juntos; se puede empezar por cualquier nodo para desplazarnos por la lista y en cualquier dirección; pueden ser vistas como listas sin comienzo ni fin.

En Python, la lista se escribe como una lista de valores (no necesariamente del mismo tipo) separados por comas y entre corchetes.

```
>>> edicion = ['butterfly', 'cobra', 'champagne', 'elvis', 'narciso']
>>> edicion
['butterfly', 'cobra', 'champagne', 'elvis', 'narciso']
```

Los índices empiezan en 0 como en las cadenas. Podemos jugar un poco con esto:

```
>>> edicion[4]
'narciso'
>>> edicion[-2]
'elvis'
>>> edicion[1:-1]
['cobra', 'champagne', 'elvis']
>>> edicion[:2] + ['hackdev','hacker']
['butterfly', 'cobra', 'hackdev', 'hacker']
```

Las cadenas son inmutables, pero en este caso en las listas se puede cambiar los elementos:

```
>>> edicion[0:1]
['butterfly']
>>> edicion[0:1] = ['primera edicion']
>>> edicion
['primera edicion', 'cobra', 'champagne', 'elvis', 'narciso']
```

Árbol

Es un conjunto de nodos conectados. El nodo es la unidad de este tipo de estructura de dato. Existe un nodo padre el cual tiene uno o más nodos hijos (rama). Solo puede haber un único nodo sin padres (que será la raíz). El nodo que no tiene hijos se conoce como hoja.

Cuando hablamos del recorrido del árbol, en una sucesión de nodos, cada dos consecutivos hay una relación de parentesco. Existen dos recorridos típicos para listar nodos: primero en profundidad, donde se listan los nodos expandiendo el hijo actual de cada nodo hasta llegar a una hoja y el primero en anchura, en el que antes de listar los nodos de nivel n+1 (a distancia n+1 aristas de la raíz) se deben haber listado los de nivel n.

Operaciones:

Las más comunes en árboles pueden ser:

- Enumerar elementos
- Buscar elementos
- Listar hijos en un nodo
- Borrar elementos
- Eliminar o añadir subárboles
- Encontrar la raíz de cualquier nodo

Entre muchas más posibles.

Tuplas y secuencias

Es un tipo de dato de secuencia, existente en Python por ser un lenguaje en evolución, la tupla consta de cierta cantidad de valores separados por comas

```
>>> h = 'hd', 'hacker', 'magazine', 'sweet'
>>> h[0]
'hd'
>>> h
('hd', 'hacker', 'magazine', 'sweet')
>>> #Para anidar tuplas
... d = h, ('exito')
>>> d
(('hd', 'hacker', 'magazine', 'sweet'), 'exito')
```

En la primera salida se encierran las tuplas entre paréntesis, esto es para que las tuplas anidadas se interpreten correctamente, en la entrada pueden ser opcionales pero es una buena práctica usarlos siempre por si la tupla que haremos se vuelva compleja.

Las tuplas son muy útiles y como las cadenas son inmutables pero se pueden crear las que contengan objetos mutables.

> *Ya es difícil encontrar un error en tu código cuando tu lo estás buscando y es aún más difícil cuando ya has asumido que tu código está libre de errores. - Steve McConnell.*

Cuando escribimos h = 'hd', 'hacker', 'magazine', 'sweet' se le denomina empaquetado de tuplas (los valores se empaquetaron en la tupla h), pero también se puede realizar la operación inversa:

```
'hd', 'hacker', 'magazine', 'sweet' = h
```

Lo que se llama desempaquetado de secuencias. La condición aquí es que sea igual al número de elementos de la secuencia.

Diccionarios

En otros lenguajes de programación se les conoce como memorias asociativas o matrices asociativas. En Python son los diccionarios que se indexan mediante claves (cadenas y números) que pueden ser de cualquier tipo inmutable. Si una tupla contiene objetos mutables no se pueden usar como clave. Por ejemplo: no se pueden utilizar listas ya que estas se pueden modificar. Un diccionario debe ser pensado como un conjunto desordenado de parejas donde las operaciones principales son de almacenamiento o extracción de valores.

El método **keys()** de un objeto devuelve todas las claves utilizadas en el diccionario. Para comprobar si una clave existe se utiliza la instrucción `key in diccionario`**:**

```
>>> edad = {'asael': 25, 'roy': 24, 'carolina': 21}
>>> edad['milale'] = 21
>>> edad
{'roy': 24, 'asael': 25, 'milale': 21, 'carolina': 21}
>>> edad['roy']
24
>>> del edad['carolina']
>>> edad
{'roy': 24, 'asael': 25, 'milale': 21}
```

> *Tener presentes las estructuras de datos en nuestro lenguaje preferido es muy importante, ya que el manejo de nuestro conjunto de datos se hará de la mejor manera posible*

Cuando uno programa debe sentirse cómodo con lo que 'codea', analizando bien cuál es la estructura que permita el manejo más adecuado del *dataset*.

Es un pájaro, es un avión ¡No! ¡Es PyOpenCL!

PYTHON

En este artículo se tratará de explicar cómo utilizar este fantástico *framework* llamado OpenCL™ para el desarrollo de aplicaciones paralelas. Nuestra aplicación desarrollada en OpenCL™ podrá correr en una amplia gama de hardware, desde súper computadoras hasta modernas tostadoras. Primero pasaremos por los conceptos básicos y luego realizaremos un ejemplo implementando un mapa logístico.

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

Licenciada en Informática (UNPSJB), actualmente realizando **Doctorado en 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**

OpenCL™ (Open Computing Language) es el primer *framework* abierto, libre de regalías, que funciona en todas las plataformas para la programación de aplicaciones que utilicen máquinas compuestas por CPUs, GPUs y otras familias de procesadores. Con OpenCL™ podemos escribir un único programa que puede correr en un gran rango de sistemas, desde celulares y *notebooks* hasta súper computadoras.

OpenCL™ mejora la velocidad y sensibilidad en un amplio espectro de aplicaciones, desde juegos y entretenimiento hasta aplicaciones científicas y médicas.

¿Cómo está especificado OpenCL™ ?

Podemos resumirlo en cuatro modelos:

1. De Ejecución: nos indica que tenemos un procesador (*host*) encargado de la coordinación de la ejecución y varios procesadores (dispositivos) encargados de ejecutar el código OpenCL C, al que llamaremos *kernels.*
2. De Plataforma: define cómo configurar el entorno del *host* y cómo los *kernels* son ejecutados en sus respectivos dispositivos, lo cual incluye la configuración del contexto en el *host* y cómo deben interactuar entre el host y los dispositivos.
3. De Memoria: nos indica cómo será la jerarquía de memoria que utilizarán los *kernels,* dado que tenemos memoria privada para cada ítem dentro de un *workgroup* y dentro de éste tendremos otra clase de memoria que será compartida por los ítem del mismo *workgroup.* OpenCL verá cómo mapear esta abstracción en el hardware real.
4. De Programación: aquí veremos cómo se mapea nuestra aplicación en el hardware.

Tratar de explicar la estructura de OpenCL™ en un artículo no sería del todo óptimo por lo que queda al lector mirar algunas de las referencias citadas al final las cuáles le darán un gran panorama sobre la teoría de programación paralela con OpenCL.

Encendiendo motores ...

Considerando que ya tienen OpenCL™ instalado es sus respectivas distribuciones solo nos encontramos a un *pip* de PyOpenCL (también pueden encontrar su repositorio en https://github.com/inducer/pyopencl.git)

Teniendo todo listo veamos cómo será la estructura de nuestra aplicación. Ésta contará de dos archivos:

- Nuestro **`main.py`**, que será alojado en el ***host*** y se encargará de crear los **contextos**[8], ***buffers***[9] necesarios, **colas**[10] y enviar a ejecución a los ***kernels***.
- Nuestro **`kernel.cl`** que tendrá el código que se ejecutará en cada dispositivo.

- Normalmente el archivo **`main.py`**, tendrá los siguientes bloques (más adelante se puede observar el código completo).

  1. Ante todo se crea el ***contexto*** y la ***cola***.

```
ctx = cl.create_some_context()
queue = cl.CommandQueue(ctx)
```

  2. Se carga el archivo .cl.

8 Un contexto necesita una plataforma más uno o varios dispositivos y se utiliza para crear las colas de ejecución. Éstas son las estructuras que permiten al *host* mandar los *kernels* a los dispositivos.
9 Los *buffers* se utilizan para guardar los datos que serán utilizados por los *kernels* como entrada y salida.
10 Las colas son las encargadas de recepción y envío de *kernels* a los distintos dispositivos.

```
program = cl.Program(ctx, cadenaConElContenidodeCL).build()
```

3. Se inicializan las variables que se encuentran en el ***host*** y luego se crean los ***buffers*** que utilizaremos para entrada y salida de los ***kernels***.

```
# Se inicializan las variables del lado del CPU
alpha = numpy.float32(valorAlpha)
output = numpy.zeros(100, numpy.float32)

# Creamos los buffers con las banderas que deseamos.
alpha_buf = cl.Buffer(ctx, mf.READ_WRITE | mf.COPY_HOST_PTR,
    hostbuf=alpha)
dest_buf = cl.Buffer(ctx, mf.READ_WRITE, output.nbytes)
```

4. Por último se ejecuta el ***kernel*** y luego se toman sus resultados.

```
 program.verhulst(queue, output.shape, (50,), self.alpha_buf,
    ..., self.dest_buf)
cl.enqueue_copy(queue, output, dest_buf)
```

Experimento Mapa Logístico

Ante todo, lo mejor sería saber qué es un mapa logístico. El mapa logístico es una formación normalizada de la ecuación de Verhulst, que nos permite la simulación del crecimiento de una población Y

Este mapa puede representarse matemáticamente como: $X_{n+1}=\alpha X_n(1-X_n)$, donde:

- α es un valor que nos indica la relación entre la natalidad y mortalidad de la población que estamos simulando.
- X_n es el tamaño de la población en el momento n

Podemos ver que variando el α los resultados son muy distintos.

Y ahora el código fuente completo de nuestro **`main.py`**:

```
# Ian Johnson.
# http://documen.tician.de/pyopencl/

import pyopencl as cl
import numpy
import pylab
import argparse
```

```
class CL:
    def __init__(self):
        self.ctx = cl.create_some_context()
        self.queue = cl.CommandQueue(self.ctx)

    def loadProgram(self, filename):
        f = open(filename, 'r')
        fstr = "".join(f.readlines())
        print fstr
        self.program = cl.Program(self.ctx, fstr).build()

    def logisticMap(self, alpha=0.9, xd=0.2):
        mf = cl.mem_flags
        self.alpha = numpy.float32(alpha)
        self.xd = numpy.float32(xd)
        self.output = numpy.zeros(100, numpy.float32)
        self.alpha_buf = cl.Buffer(self.ctx, mf.READ_WRITE | mf.COPY_HOST_PTR,
                                   hostbuf=self.alpha)
        self.xd_buf = cl.Buffer(self.ctx, mf.READ_WRITE | mf.COPY_HOST_PTR,
                                self.xd.nbytes, hostbuf=self.xd)
        self.dest_buf = cl.Buffer(self.ctx, mf.READ_WRITE, self.output.nbytes)

    def execute(self):
        self.program.verhulst(self.queue, self.output.shape, (50,),
            self.alpha_buf, self.xd_buf, self.dest_buf)
        cl.enqueue_copy(self.queue, self.output, self.dest_buf)
        print "alpha: ", self.alpha
        print "xn: ", self.xd
        print "results: ", self.output


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description=
                                     'Logistic Map with control parameter')
    parser.add_argument("--alpha", dest="alpha", default = 0.9,
        help='This option is used to set the alpha')
    args = parser.parse_args()
    alphaIn = args.alpha

    myLogisticMap = CL()
    myLogisticMap.loadProgram("verhulst.cl")
    myLogisticMap.logisticMap(alpha=alphaIn)
    myLogisticMap.execute()

    pylab.plot(myLogisticMap.output, 'ro-')
    pylab.ylim(0, 1)
    pylab.text(40, 0.2, r'$\alpha = %f$' % (myLogisticMap.alpha))
    pylab.title("Successive Iterations of the logistic Map")
    pylab.show()
```

Y el código del *kernel* ***verhulst.cl***:

```
__kernel void verhulst(global float* alpha, global float* xd, global float* output)
{
    float temp;
    int idx = get_local_id(0)+get_local_size(0)*get_group_id(0);
    temp = xd[0];
    int i;
    for(i=0;i<=idx;i++)
```

```
    {
        temp = alpha[0]*temp*(1-temp);
    }
    output[idx] = temp;
}
```

Referencias y Links Interesantes

[1] Aaftab Munshi, Benedict Gaster, Timothy Mattson, James Fung, Dan Ginsburg. **OpenCL Programming Guide,** Addison-Wesley (2011).

[2] Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, Dana Schaa. **Heterogeneous Computing with OpenCL**, Morgan Kaufmann (2011).

[3] http://mathema.tician.de/software/pyopencl

[4] http://gpgpu2.blogspot.com.ar/2012/09/efficient-convolution-on-multi.html

Unit Testing con PHPUnit y PyUnit

En la edición Nº3 de Hackers & Developers Magazine hicimos una introducción al desarrollo dirigido por pruebas, prometiendo hablar de PyUnit y PHPUnit en una siguiente entrega. Como lo prometido es deuda, aquí está TDD con Python y PHP.

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**

Pruebas Unitarias en PHP

Existen varios *frameworks* xUnit para Unit Testing en PHP, pero sin dudas, el único que ha demostrado contar con una gran cobertura de código, estabilidad y buena documentación, es PHPUnit. El manual oficial de PHPUnit (en inglés) puede encontrarse en: http://www.phpunit.de/manual/current/en/index.html

Se puede instalar PHPUnit en sistemas operativos GNU/Linux, mediante PEAR:

```
sudo pear upgrade PEAR
pear config-set auto_discover 1
pear install pear.phpunit.de/PHPUnit
```

Aunque en distribuciones basadas en Debian, también puede hacerse directamente mediante la instalación del paquete `phpunit` con `apt-get`:

```
sudo apt-get install phpunit
```

Métodos Assert de PHPUnit

PHPUnit provee una gran cantidad de métodos `assert`, cuyas referencias podemos encontrar en el Capítulo 4 del manual oficial:

http://www.phpunit.de/manual/3.6/en/writing-tests-for-phpunit.html

Algunas características comunes de los métodos `assert`, son:

- Generalmente, por cada método `assert` existe su opuesto: `assertContains()` y `assertNotContains()`.
- A la vez, cada método `assert` deberá recibir mínimamente un parámetro que será el resultado de ejecutar el código del SUT.
- Adicionalmente, a cada método `assert`, se le puede pasar como parámetro opcional, un mensaje personalizado para ser arrojado en caso de error (generalmente, será el último parámetro).
- Los métodos `assert` que requieren el paso de dos parámetros obligatorios (valores que deben compararse entre sí), generalmente guardan el siguiente orden:

```
metodoAssert($valor_esperado, $valor_recibido)
```

Es decir, que en esos casos, siempre el primer parámetro será el valor esperado y el segundo parámetro, el valor recibido por la ejecución del código SUT.

Veamos algunos ejemplos puntuales:

```
<?php
# Archivo TestCase: BalanceContable_Test.php

require_once 'BalanceContable.php';  # SUT

class BalanceContableTest extends PHPUnit_Framework_TestCase {

    public function setUp() {
        $this->coverage = new BalanceContable();
        $this->coverage->alicuota_iva = 21;
    }

    // AssertEquals($valor_esperado, $valor_recibido)
    public function test_calcular_iva() {
        $this->coverage->importe_bruto = 1500;
        $result = $this->coverage->calcular_iva();
        $this->assertEquals(315, $result);
    }

    // AssertTrue($valor_recibido)
    public function test_alcanzado_por_impuesto_de_importacion_con_160() {
        $this->coverage->importe_bruto = 160;
```

```
        $result = $this->coverage->alcanzado_por_impuesto_de_importacion();
        $this->assertTrue($result);
    }

    // AssertNull($valor_recibido)
    public function test_alcanzado_por_impuesto_de_importacion_con_143() {
        $this->coverage->importe_bruto = 143;
        $result = $this->coverage->alcanzado_por_impuesto_de_importacion();
        $this->assertNull($result);
    }
}

?>


# Código SUT: BalanceContable.php
<?php

class BalanceContable {

    public $importe_bruto;
    public $alicuota_iva;

    # Calcular IVA sobre un importe bruto
    public function calcular_iva() {
        $iva = $this->alicuota_iva / 100;
        $neto = $this->importe_bruto * $iva;
        return $neto;
    }

    # Determinar si un importe paga impuesto de importación
    public function alcanzado_por_impuesto_de_importacion() {
        // importes mayores a 150 USD pagan impuesto
        if($this->importe_bruto > 150) {
            return True;
        }
    }

}

?>
```

Pruebas Unitarias en Python

PyUnit es el *framework* xUnit elegido de forma oficial por Python desde su versión 1.5.2. Si bien existen muchos otros, generalmente están destinados a ampliar los beneficios de PyUnit para la realización de test más complejos, como el caso de PyDoubles[11] -creado por Carlos Blé-. Toda la referencia sobre PyUnit se encuentra en el manual oficial de Python (en inglés): http://docs.python.org/library/unittest.html

PyUnit no necesita ser instalado ya que desde la versión 2.1 forma parte de la librería estándar de Python, a través del módulo unittest.

11 http://www.carlosble.com/category/software-development/pydoubles/

Métodos Assert de PyUnit

Una lista completa de los métodos assert de PyUnit puede encontrarse en la documentación sobre `unittest` de Python en la siguiente URL:

http://docs.python.org/library/unittest.html#assert-methods

Una diferencia particular que existe entre PyUnit y otros frameworks como PHPUnit, es que nos permite efectuar afirmaciones, con una sintaxis bastante simple, sin necesidad de recurrir a métodos `assert` específicos:

```
assert resultado == valor_esperado
```

En un ejemplo más concreto, podríamos verlo así (donde *coverage* será la instancia al objeto del SUT):

```
assert self.coverage.sumar_dos_numeros(5, 15) == 20
```

Otra diferencia fundamental con PHPUnit, es que el método `assert`<*Igualdad*>, posee su nombre en singular:

```
PHPUnit:
    assertEquals($a, $b);

PyUnit:
    asserEqual(a, b)
```

Algunas características comunes de los métodos `assert`, son:

- Al igual que con PHPUnit, generalmente, por cada método `assert` existe su opuesto: `assertEqual()` y `assertNotEqual()`.
- También, siguiendo nuevamente la línea de PHPUnit, cada método `assert` deberá recibir mínimamente un parámetro que será el resultado de ejecutar el código del SUT y opcionalmente, como último parámetro, puede recibir un mensaje personalizado para ser arrojado en caso de error.
- A diferencia de PHPUnit, los métodos `assert` que requieren el paso de dos parámetros obligatorios (valores que deben compararse entre sí), generalmente guardan el siguiente orden:

```
metodoAssert(valor_recibido, valor_esperado)
```

Es decir, que en esos casos, siempre el primer parámetro será el valor recibido

por la ejecución del código SUT y el segundo parámetro, el valor esperado.

Veamos el ejemplo realizado anteriormente en PHP, pero esta vez, en Python con PyUnit:

```
# -*- coding: utf-8 -*-
# Archivo TestCase: test_balance_contable.php
import unittest
from balance_contable import BalanceContable


class BalanceContableTestCase(unittest.TestCase):

    # setUp()
    def setUp(self):
        self.coverage = BalanceContable()
        self.coverage.alicuota_iva = 21

    # assertEqual(valor_recibido, valor_esperado)
    def test_calcular_iva(self):
        self.coverage.importe_bruto = 2500
        result = self.coverage.calcular_iva()
        self.assertEqual(result, 525)

    # AssertTrue(valor_recibido)
    def test_alcanzado_por_impuesto_de_importacion_con_160(self):
        self.coverage.importe_bruto = 160
        result = self.coverage.alcanzado_por_impuesto_de_importacion()
        self.assertTrue(result)

    # AssertIsNone(valor_recibido)
    def test_alcanzado_por_impuesto_de_importacion_con_143(self):
        self.coverage.importe_bruto = 143
        result = self.coverage.alcanzado_por_impuesto_de_importacion()
        self.assertIsNone(result)


# Necesario para correr los test si es llamado por línea de comandos
if __name__ == "__main__":
    unittest.main()
```

Nótese que el método assertEquals de PHPUnit, se denomina assertEqual (en singular) en PyUnit y que en reemplazo del método assertNull, PyUnit propone assertIsNone (esto es debido a que Python no retorna valores nulos como tales, sino como “None”).

```
# -*- coding: utf-8 *-*
# Código SUT: balance_contable.php


class BalanceContable(object):

    def __init__(self):
        self.importe_bruto = 0
        self.alicuota_iva = 0
```

```
    # Calcular IVA sobre un importe bruto
    def calcular_iva(self):
        iva = self.importe_bruto * self.alicuota_iva / 100
        return iva

    # Determinar si un importe paga impuesto de importación
    def alcanzado_por_impuesto_de_importacion(self):
        # importes mayores a 150 USD pagan impuesto
        if self.importe_bruto > 150:
            return True
```

"Descubriendo" Test en Python

Desde la versión 2.7 de Python, ya no es necesario realizar "maniobras" o crear Test Suites, con el único fin de correr todos los Test de nuestra aplicación, de un solo paso. Todos los test de una aplicación, pueden correrse mediante el comando `discover`:

```
eugenia@cocochito:~/proyectos$ python -m unittest discover
```

`discover`, "descubrirá" todos los test, identificándolos por el nombre del archivo: debe comenzar por el prefijo "test" (`discover` utiliza la expresión regular test*.py para identificar Test Cases). Además, debe tenerse en cuenta que el nombre de los métodos de prueba, también deben comenzar por el prefijo test.

Sin embargo, podría pretender ejecutarse solo un TestCase:

```
eugenia@cocochito:~/proyectos$ python test_balance_contable.py
```

O un test en particular de una clase TestCase:

```
eugenia@cocochito:~/proyectos$ python test_balance_contable.py
BalanceContableTestCase.test_calcular_iva
```

También es posible, pasar el parámetro `-v` a fin de obtener un reporte más detallado:

```
eugenia@cocochito:~/proyectos$ python -m unittest discover -v
test_alcanzado_por_impuesto_de_importacion_con_143
(Test.test_balance_contable.BalanceContableTestCase) ... ok
test_alcanzado_por_impuesto_de_importacion_con_160
(Test.test_balance_contable.BalanceContableTestCase) ... ok
test_calcular_iva (Test.test_balance_contable.BalanceContableTestCase) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Manual de Perl (Parte III)

Una de las características más importantes de Perl es, quizás, su potencia con la expresiones regulares. En esta entrega veremos cómo funcionan.

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**

Una expresión regular es un patrón que describe un conjunto de cadenas. Las expresiones regulares se construyen combinándolas con ciertos operadores para aumentar la complejidad.

Podemos distinguir tres usos de expresiones regulares:

Comparación

Nos permiten saber si un patrón se encuentra dentro de una cadena. Su sintaxis es la siguiente:

```
$var =~ /patrón/;
```

Para la variable **$_** (la variable por defecto), podemos poner directamente la expresión regular:

```
/patrón/;
```

Como podemos ver, los patrones se encierran entre (**/**). En el caso de querer conocer si no se encuentra, utilizaremos el operador **!~**.

Un ejemplo:

```
$cadena = "Esto es una prueba";

if ( $cadena =~ /prueba/ ) {
   print "El patrón se encuentra en la cadena\n";
}
if ( $cadena != /prueba/ ) {
   print "El patrón no se encuentra en la cadena\n";
}
```

En Perl, el operador para indicar que vamos a aplicar una expresión regular es **=~**. El patrón debe ir entre (**/**):

```
$cadena =~ /patrón/;
Buscaría si existe patrón dentro de $cadena.
```

El operador **!=** devuelve falso si el patrón se encuentra en la cadena. Su significado vendría a ser: Si cadena **no contiene** patrón.

En combinación con los caracteres que queramos buscar, podemos utilizar combinaciones con metacaracteres. Con ellos podremos indicar cuántas veces queremos que se repita un carácter, utilizar comodines, buscar al principio o final de la cadena, etc.

Vamos a verlos con un ejemplo:

```
$cad = 'cometa cometas 1cometa metas';

# \d    Busca un dígito. En mayúscula, \D, un carácter no dígito.
   if ($cad =~ /\d/) {      #resultado:
      print "$`<$&>$'\n";   #cometa cometas <1>cometa metas
   }

# \w   Un carácter alfanumérico. En mayúscula, \W, un carácter no alfanumérico.
   if ($cad =~ /\w/) {        #Resultado:
      print "$`<$&>$'\n";     #<c>ometa cometas 1cometa metas
   }
# .   Cualquier carácter que no sea retorno de carro
   if ($cad =~ / .cometa/) {   #Resultado:
      print "$`<$&>$'\n";      #cometa cometas< lcometa> metas
   }

# \s   Un carácter de espaciado (espacio en blanco, salto de línea, tabulador,
# retorno de carro). En mayúscula significa lo contrario.
   if ($cad =~ /\s/) {         #Resultado:
      print "$`<$&>$'\n";      #cometa< >cometas 1cometa metas
   }
```

He utilizado unas variables especiales, su contenido es:

```
$`  La parte de la cadena anterior al inicio del patrón encontrado.

$&  El patrón.

$'  El resto de la cadena.
```

En ocasiones nos interesa buscar listas o rangos de caracteres. Para esto utilizaremos los corchetes []. Unos ejemplos de rangos:

```
1.- Letras minúsculas: a-z
2.- Letras mayúsculas: A-Z
3.- Todas las letras: a-zA-Z
4.- Dígitos: 0-9
```

Utilización de los []:

```
# [] Para buscar listas o rangos de caracteres.
   if ($cad =~ /[aeiou]/) {    #Busca una vocal. Resultado:
      print "$`<$&>$'\n";      #c<o>meta cometas 1cometa metas
   }
```

Otra de las cosas que podemos hacer es añadir cuántas veces queremos que se repita un carácter o una lista y también especificar límites de palabra o cadena:

```
# ^ Debe coincidir con el principio de la cadena
   if ($cad =~ /^co/) {        #¿Empieza la cadena con co?. Resultado:
      print "$`<$&>$'\n";      #<co>meta cometas 1cometa metas
   }

# b límites de una palabra
   if ($cad =~ /\bco/) {       #Palabras que empiecen con co. Resultado:
      print "$`<$&>$'\n";      #<co>meta cometas 1cometa metas
   }
   if ($cad =~ /ta\b/) {       #Palabras que terminen con ta. Resultado:
      print "$`<$&>$'\n";      #come<ta> cometas 1cometa metas
   }

# B interior de una palabra
   if ($cad =~ /\Bco/) {       #Palabras que contengan co en el principio:
      print "$`<$&>$'\n";      #cometa cometas 1<co>meta metas
   }
   if ($cad =~ /ta\B/) {       #Palabras que contengan ta al final:
      print "$`<$&>$'\n";      #cometa come<ta>s 1cosmeta metas
   }

# {n,m} Número de veces, mínimo y máximo, que se debe repetir un carácter.
# {n,}  Número mínimo de veces, sin límite superior.
# {n}   Número exacto de veces.
# *     Es equivalente a {0,}, se repite 0 más veces.
# +     Es equivalente a {1,}, se repite al menos una vez.
# ?     Es equivalente a {0,1}, aparece, cómo mucho, una vez.
```

```
$cad = "cometa cometas 1cococometata cococometas";
   if ($cad =~ /\b(co){2,3}/) {    #Que empiece con co, repetida min 2, max 3 veces
      print "$`<$&>$'\n";          #cometa cometas 1cococometata <cococo>metas
   }
   if ($cad =~ /(ta){2,3}\b/) {    #Que termine con ta,repetida min 2, max 3 veces:
      print "$`<$&>$'\n";          #cometa cometas 1cococome<tata> cococometas
   }

# Operador | (or). Establece alternativas:
   if ($cad =~ /(\Bco)|(ta\B)/) {     #Contiene co al principio o ta al final:
      print "$`<$&>$'\n";             #cometa come<ta>s 1cometa metas
   }
```

El símbolo (^) tiene dos significados. Uno es indicar principio de cadena pero si aparece dentro de los corchetes, [^conjunto], significa que **no** exista ese conjunto de caracteres en la cadena.

Sustitución

Nos permite, una vez encontrado un patrón, sustituirlo por una cadena. Para ello se utiliza la siguiente sintaxis:

```
s/patrón/cadena/opciones
```

Ejemplo:

```
$cad = "cometa cometas 1cococometata cococometas";
$cad =~ s/(co){2,3}/co/gi;
print "El resultado es $cad";

# $cad contiene ahora: cometa cometas 1cometata cometas
```

Las opciones que disponemos:

```
-g Por defecto, en la primera ocurrencia del patrón se para la búsqueda y
sustitución. Esta opción es para que busque todas las ocurrencias.

-i Ignora mayúsculas y minúsculas.

-e evalúa la expresión.
```

Translación

Reemplaza cada carácter de la lista inicial con los caracteres de la lista final. Devuelve el número de intercambios realizados. Un ejemplo:

```
#Transforma una cadena a mayúsculas:
$cad =~ tr/a-z/A-Z/;

#contar cuántas C aparecen:
$n =$cad =~ tr/C/C);
```

Recordando patrones

Disponemos de unas variables especiales para referenciar a los patrones. Los patrones podemos agruparlos entre paréntesis, de forma que cada variable hará referencia a un grupo.

Para utilizar fuera de la expresión, disponemos de las variables *$1, $2…$n,* siendo *n* el número de grupos. Dentro de la expresión, podemos utilizar *\1,\2...\n.*

Ejemplo: Vamos a buscar las palabras que empiecen por **CO** y contengan **TA** , después, vamos a intercambiar esos valores para todas las ocurrencias e ignorando mayúsculas/minúsculas:

```
$cad = “COMETA COMETAS”;

$cad =~ s/(co)(\w*)(ta)/\3\2\1/gi;
print “$cad”;
#resultado: TAMECO TAMECOS
```

Con esto concluye esta entrega. En la próxima os hablaré de cómo utilizar los ficheros y las funciones.

Enlaces de interés:

http://www.perl.com/

http://www.cpan.org/

Configurando GIT en Ubuntu Server

Hace mucho tiempo, en mi trabajo, teníamos el grave problema de no poder manejar las versiones de los desarrollos. No faltaba ese día apocalíptico en el que técnicamente perdías más de un mes de trabajo, porque a alguien se le ocurría subir sus cambios y los tuyos, simplemente, quedaban en el olvido. Esto se arregló con el control de versiones y aquí les mostraré cómo, en un servidor Ubuntu, configurar GIT: uno de los sistemas de control de versiones más usados, creado por nada más y nada menos que Linus Torvalds.

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

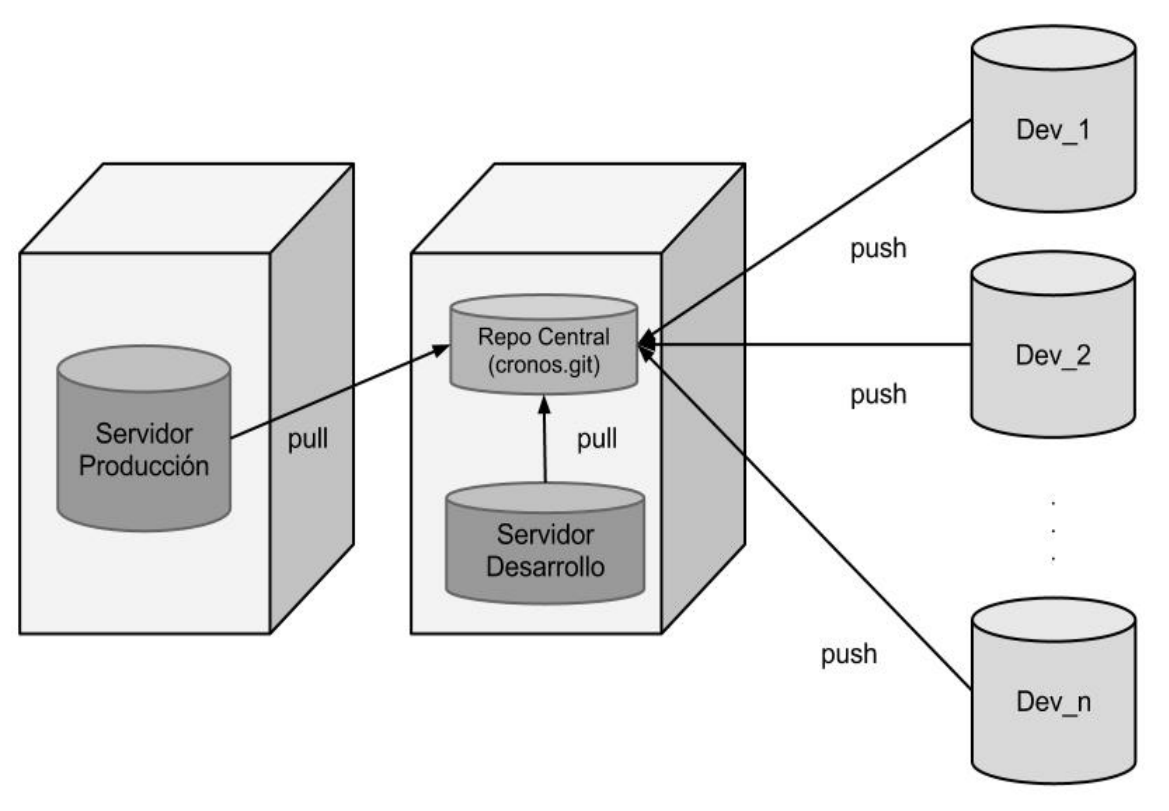
Redes sociales:
Twitter: **@indirabm**

Git es un software de control de versiones diseñado por Linus Torvalds, pero ¿Qué hace diferente a GIT de otros sistemas similares?

GIT es un sistema distribuido. Esto significa que existen repositorios en cada uno de los usuarios (desarrolladores). Personalmente, considero que eso hace que sea mucho mas fácil su trabajo, debido a que se pueden hacer muchos *commits* locales (agrupar cambios y guardarlos) antes de enviar los cambios al servidor de desarrollo o de producción (push). Esto es genial dado que mis cambios solo afectan a mi repositorio local. Si el cambio que se desea hacer, es lo suficientemente importante, se puede crear un nuevo branch (rama donde puedo hacer cambios mayores). Esto no pasaba con los sistemas centralizados en los cuales el repositorio estaba en un solo lugar (servidor) y por lo tanto, cuando hacías un *commit* tenías que cerciorarte de que los nuevos cambios no "rompieran" los de otros.

Ahora que ya sabemos qué es GIT vamos a implementarlo. Primero que todo es necesario entender cómo trabaja GIT y saber cómo es la distribución.

Como se puede ver en la imagen de abajo, GIT aísla todos los repositorios de tal manera que cada uno pueda trabajar de forma independiente. Además tenemos que contar con un repositorio central, el cual tendrá todos los cambios realizados. Por supuesto, cada desarrollador podría trabajar en una rama (branch). Usualmente, el servidor de producción tiene su propia rama así como el de desarrollo.



Configuración de un repositorio central (servidor)

Vamos a configurar un repositorio central y dos repositorios locales para los desarrolladores que trabajen en él.

```
Instalar GIT
root@dev:~# sudo apt-get install git

Configuración de usuarios
root@dev:~# git config --global user.name "Argos"
root@dev:~# git config --global user.email argos@hd.com

Instalación de openssh (necesario para transferir datos mediante el protocolo SSH)
root@dev:~# apt-get install openssh-server

Creación del usuario "git" (propietario del repositorio)
root@dev:~# adduser git
```

Configuramos el servidor para autenticarnos mediante llaves públicas en vez de hacerlo con contraseñas.

```
Abrimos el archivo "sshd_config"
root@dev:~# nano /etc/ssh/sshd_config

Agregamos las siguientes líneas:
PubkeyAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys
```

Ahora, creamos las llaves públicas:

```
Cambiamos al usuario "git" que creamos anteriormente
root@dev:~# su git

Nos movemos al "home" del usuario
git@dev:/home/argos$ cd ~

Creamos la carpeta oculta "ssh" para almacenar las llaves públicas
git@dev:~$ mkdir .ssh

Creamos el archivo "authorized_keys":
git@dev:~$ touch .ssh/authorized_keys

Creamos una carpeta para guardar las llaves públicas que nos envíen, así tendremos
un mejor control de quienes se están comunicando con nuestro servidor.
git@dev:~$ mkdir desarrolladores
```

Finalmente, crearemos un directorio que almacenará nuestro repositorio:

```
git@dev:~$ mkdir dev
git@dev:~$ cd dev
```

A continuación, inicializamos nuestro primer proyecto, al que llamaremos “cronos”:

```
git@dev:~/dev$ mkdir cronos.git
git@dev:~/dev/cronos.git$ cd cronos.git
git@dev:~/dev/cronos.git$ git --bare init
```

Hemos inicializado un repositorio vacío que se utlizará como repositorio central. Aquí encontrarán todos los *commit* que vayan realizando al proyecto, sin embargo este repositorio no se usa para desarrollar.

Configuración de los repositorios locales

Las siguientes instrucciones deberán seguirse en el ordenador de cada uno de los desarrolladores

El primer paso, será instalar y configurar GIT.

```
root@jose:~# apt-get install git
root@jose:~# git config --global user.name "Jose"
root@jose:~# git config --global user.email jose@hd.com
```

A continuación, generamos la llave pública y la enviamos al servidor:

```
root@jose:~# ssh-keygen -t rsa
```

Nos pide una frase clave para usar la llave, esta tiene que ser mayor a 4 dígitos. Luego, enviamos la llave pública a la carpeta `desarrolladores` en nuestro servidor.

```
root@jose:~# scp /root/.ssh/id_rsa.pub git@192.168.1.9:desarrolladores/jose_rsa.pub
```

A continuación, en el servidor, agregaremos la llave enviada a `authorized_keys`:

```
git@dev:~$ cat desarrolladores/jose_rsa.pub >> .ssh/authorized_keys
```

Nuevamente, en el ordenador del desarrollador, crearemos un directorio para el proyecto:

```
root@jose:~# mkdir cronos
root@jose:~# cd cronos/

Ahora agregaremos un archivo como prueba
root@jose:~/cronos# echo "hola HD :P " >> README

Inicializamos git
root@jose:~/cronos# git init

Agregar todos los archivos que se desea enviar al servidor
También se puede enviar archivo por archivo o por tipo de archivos.
root@jose:~/cronos# git add .

Enviamos nuestro primer commit
root@jose:~/cronos# git commit -m "Prueba del primer commit"

Agregamos la ruta del repositorio central
```

```
root@jose:~/cronos# git remote add origin git@192.168.1.9:~/dev/cronos.git

Enviamos los cambios al repositorio central
root@jose:~/cronos# git push origin master
Le pedirá que ingrese la frase con la que creo la clave pública
```

Para que otros desarrolladores puedan recuperar los cambios enviados, se utiliza el comando “pull” (en vez de push):

```
root@jose:~/cronos# git pull origin master
```

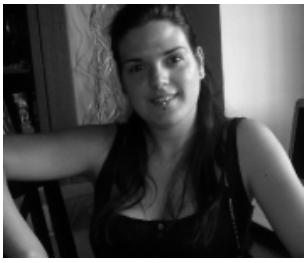
Para visualizar el historial de cambios se puede utilizar una GUI como GitWeb o sino, por línea de comandos mediante `git log`:

```
root@jose:~/cronos# git log --stat
```

Pásate a GNU/Linux con Arch Linux: ¿Qué es ABS? y... ¿AUR?

Una de las cosas que hacen Arch especial es su sistema de ports, ABS y el «AUR», el repositorio comunitario, donde podemos encontrar multitud de paquetes e incluso subir aquellos que creemos nosotros. Veamos cómo utilizarlos.

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**

En Arch, además de los repositorios con los paquetes ya compilados, disponemos de un tipo *ports* similar al de los sistemas *BSD: el ABS. Basándose en ABS, Arch ofrece un repositorio comunitario, AUR, donde poder subir, votar y encontrar software que aún no ha pasado a los repositorios.

¿Qué son los ports? ¿Qué es ABS?

Un *port* no es más que una carpeta con el nombre del software a instalar, que contiene algunos archivos o *scripts* con las instrucciones necesarias para construir el software desde el código fuente. Así, en estos sistemas, ejecutando un sólo comando, **make install**, instalamos el software.

El ABS es un árbol de carpetas, que se encuentra en `/var/abs`, donde estarán todos los `PKGBUILD` correspondientes a los paquetes que se encuentran compilados en los repositorios. Dentro de `/var/abs` encontramos las carpetas correspondientes a cada repositorio (`core`, `community`, `extra`…). Dentro de cada uno, las carpetas correspondientes a los nombres de los paquetes respectivos que haya en ese repositorio.

Cada carpeta contiene el `PKGBUILD` con la dirección de descarga y las instrucciones de compilación, además de otros archivos con instrucciones de instalación. Podemos editar el `PKGBUILD` y adaptarlo a nuestras necesidades.

Utilizaremos **makepkg** para crear el paquete, que será instalado con **pacman** después. Aquí es donde podemos aplicar optimizaciones de compilación.

Para que podamos utilizar ABS, debemos instalarlo:

```
# pacman -S abs
```

Para activar los repositorios que queremos estén disponibles, editaremos `/etc/abs.conf` y, en la variable `REPOS`, quitaremos (activar) o pondremos (desactivar) un (!) a cada repositorio que aparece en la lista:

```
REPOS=(core extra community multilib !testing !community-testing !multilib-testing)
```

Tanto para descargar el árbol como para actualizarlo, basta ejecutar:

```
# abs
```

Ahora vamos a crear una carpeta donde copiaremos el contenido del paquete que queremos crear. Por ejemplo:

```
$ mkdir ~/abs
$ cp -r /var/abs/core/linux ~/abs
```

Ya podemos realizar los cambios que necesitemos en el `PKGBUILD`.

Creando el paquete: makepkg

Lo primero que debemos hacer es revisar nuestro `/etc/makepkg.conf` y personalizar el campo del empaquetador. Así tendremos un control sobre qué paquetes hemos creado nosotros:

```
#-- Packager: name/email of the person or organization building packages
PACKAGER="nombre <correo@ejemplo.com>"
```

Podemos optimizar la compilación estableciendo las `CFLAGS` y `CXXFLAGS` con los valores soportados por nuestra arquitectura. Procederemos de la siguiente manera:

```
$ gcc -march=native -E -v - </dev/null 2>&1 | sed -n 's/.* -v - //p'
```

Nos devolverá algo parecido a:

```
-march=nocona -mcx16 -msahf -mno-movbe -mno-aes -mno-pclmul -mno-popcnt -mno-abm
-mno-lwp -mno-fma -mno-fma4 -mno-xop -mno-bmi -mno-bmi2 -mno-tbm -mno-avx -mno-avx2
-mno-sse4.2 -mno-sse4.1 -mno-lzcnt -mno-rdrnd -mno-f16c -mno-fsgsbase --param l1-
cache-size=16 --param l1-cache-line-size=64 --param l2-cache-size=2048
-mtune=nocona
```

Con estos datos, el valor de nuestro `CFLAGS` y `CXXFLAGS` será:

```
CFLAGS="-march=nocona -mtune=generic -O2 -pipe -mcx16 -msahf -mno-movbe -mno-aes
-mno-pclmul -mno-popcnt -mno-abm -mno-lwp -mno-fma -mno-fma4 -mno-xop -mno-bmi
-mno-bmi2 -mno-tbm -mno-avx -mno-avx2 -mno-sse4.2 -mno-sse4.1 -mno-lzcnt -mno-rdrnd
-mno-f16c -mno-fsgsbase --param l1-cache-size=16 --param l1-cache-line-size=64
--param l2-cache-size=2048”

CXXFLAGS="${CFLAGS}"
```

Es importante cambiar **`mtune`** a **`generic`** y una buena idea, añadir **`-O2 -pipe`**.

Si disponemos de un equipo con más de un núcleo, podemos modificar el valor de `MAKEFLAGS`. Comprobamos el número de núcleos con **`nproc`**, descomentamos (quitando la #) la variable `MAKEFLAGS` y establecemos su valor a `j<número_nproc>`. Por ejemplo, si **`nproc`** nos devuelve 4, debemos dejar el valor:

```
MAKEFLAGS="-j4"
```

Para agilizar las compilaciones con gcc, podemos utilizar la herramienta **ccache**:

```
# pacman -S ccache
```

Editamos `/etc/makepkg.conf` y, en la variable `BUILDENV`, eliminamos la (!) de **ccache**, quedando de forma similar a:

```
BUILDENV=(fakeroot !distcc color ccache check !sign)
```

Una vez que hemos dejado nuestro `/etc/makepkg.conf` acorde a nuestras necesidades, procedemos a crear el paquete.

Siguiendo con el ejemplo anterior, dentro de la carpeta `~/abs/linux`, ejecutamos como usuario:

```
$ makepkg -s
```

Creará un archivo con la extensión `pkg.tar.xz`, el cual instalaremos con:

```
# pacman -U <nombre_de_paquete>.pkg.tar.xz
```

Podemos hacer esto con una sola orden:

```
$ makepkg -si
```

Una vez creado, llamará a **pacman** para instalar el paquete.

NOTA: Para que un usuario pueda instalar paquetes con **makepkg** (parámetro **-s** y **-i**), es necesario tener instalado **sudo** y que nuestro usuario tenga permisos de administrador

Arch User Repository (AUR)

Según podemos leer en wiki[12]:

> *"AUR (Arch User Repository) es el lugar donde la comunidad de Arch Linux puede subir los PKGBIILD de las aplicaciones, bibliotecas, etc., y compartirlos con el resto de la comunidad. Los demás usuarios pueden votar para que sus favoritos entren en el repositorio [community], de modo que puedan ser instalados en Arch Linux en formato binario."*

En este repositorio nos vamos a encontrar un *tarball* con el PKGBUILD y demás archivos necesarios para crear un paquete. El mecanismo de instalación es sencillo: descargar el *tarball*, descomprimir, crear paquete e instalar. Por ejemplo, para instalar **package-**

12 https://wiki.archlinux.org/index.php/Arch_User_Repository_(Español)#Q:_.C2.BFQue_es_AUR.3F

query:

```
$ wget https://aur.archlinux.org/packages/pa/package-query/package-query.tar.gz
$ tar zxvf package-query.tar.gz
$ cd package-query
$ makepkg -si
$ cd ..
$ rm -r package-query
```

Disponemos de múltiples herramientas que permiten automatizar el proceso de instalación de paquetes del AUR. Una herramienta muy popular es **yaourt**, la cual también permite instalar los paquetes disponibles en los repositorios desde el código fuente (sin necesidad de tener instalado ABS).

Para instalar esta herramienta, una vez instalado el paquete **package-query**, ejecutamos:

```
$ wget https://aur.archlinux.org/packages/ya/yaourt/yaourt.tar.gz
$ tar zxvf yaourt.tar.gz
$ cd yaourt
$ makepkg -si
$ cd ..
$ rm -r yaourt
```

El trabajo con **yaourt** es similar a trabajar con **pacman.** Comentaré las nuevas opciones que incorpora:

```
yaourt <patrón>       Busca tanto en los repositorios como en el AUR aquellos
                      paquetes que cumplan el <patrón>

yaourt -Syua          Actualiza los paquetes, incluyendo los que proceden del AUR.

yaourt  -Syua --devel Actualiza el sistema completo incluyendo los paquetes
                      procedentes del AUR. Si tenemos paquetes basados en cvs,
                      svn, git, bzr,  se actualizarán a la última versión que
                      exista en sus repositorios.

yaourt -Sb <package>  Compila <package> perteneciente a los repositorios desde las
                      fuentes (no es necesario tener instalado ABS)

yaourt -C             Nos muestra la lista de archivos *.pac* (copias de seguridad
                      de determinados archivos que se crean al actualizar el
                      sistema) y nos permite editarlos y eliminarlos.

yaourt -B             Realiza una copia de seguridad de la BBDD de pacman en la
                      carpeta raíz de nuestro usuario.
```

En el archivo de configuración de **yaourt**, `/etc/yaourtrc` es conveniente cambiar la carpeta de archivos temporales. Por defecto apunta a `/tmp`, pero `/tmp` está montado como **tmpfs** (es decir, está montado en memoria RAM). Al descargar, compilar, etc. se ocupa bastante espacio y podemos tener problemas, bien de lentitud de nuestro equipo, bien porque se quede sin espacio y no se pueda crear el paquete.

Podemos crear una carpeta, por ejemplo `/home/yaourt`, y establecer ésta cómo la carpeta temporal:

```
# mkdir /home/yaourt
# chmod 775 /home/yaourt
# chown root:users /home/yaourt
```

Disponemos de dos métodos para modificar la configuración:

1.- Estableciendo la configuración para cada usuario:

```
$ echo TMPDIR="/home/yaourt" > ~/.yaourtrc
```

2.- Establecer el valor en `/etc/yaourtrc`, para que sea esa la carpeta por defecto para todos los usuarios:

Localizamos **TMPDIR**, eliminamos la (#) en caso de tenerla y establecemos el valor:

```
TMPDIR=/home/yaourt
```

Nota final

Gracias a ABS, disponemos de una herramienta que nos permite personalizar aún más nuestro sistema. Gracias a él disponemos de AUR, un gran repositorio con multitud de paquetes.

En general, las versiones beta o las versiones de desarrollo de una buena parte de los paquetes oficiales, podemos encontrarlas allí.

También podemos encontrar paquetes configurados de forma distinta (por ejemplo, con soporte para **konsole-kit** en lugar de **logind**) y multitud de aplicaciones que aún no han subido a *[community]*, dándonos la posibilidad de votar aquellos paquetes que nos gustaría estuviesen disponibles en *[community]*, lo que permite que la comunidad tenga una participación activa con el desarrollo de la distribución.

Debemos tener en cuenta que los paquetes de AUR no están soportados oficialmente. Al instalar utilizando alguna herramienta como **yaourt**, **packer**, **pacaur**, **aura**... se nos permite editar el PKGBUILD. Es conveniente revisarlo, para comprobar de dónde se descargan los paquetes o qué parches se aplican.

Aunque los **Trusted User (TU)**[13] se encargan de comprobar que los paquetes no incluyen código malicioso, están bien formados, etc. para subir un paquete al AUR sólo debemos registrarnos en la página y subir el archivo creado con **makepkg --source**, con lo que, en principio, debemos ser precavidos.

Enlaces de interés:

https://wiki.archlinux.org/index.php/Arch_Build_System

https://wiki.archlinux.org/index.php/Pkgbuild

https://wiki.archlinux.org/index.php/AUR

https://wiki.archlinux.org/index.php/Makepkg

https://wiki.archlinux.org/index.php/Yaourt

http://en.gentoo-wiki.com/wiki/Hardware_CFLAGS

13 Thusted User (TU): Son los usuarios de confianza, se encargan de mantener los paquetes de [community] y de supervisar AUR.

¿Cómo son los Equipos Ágiles? (Parte I)

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

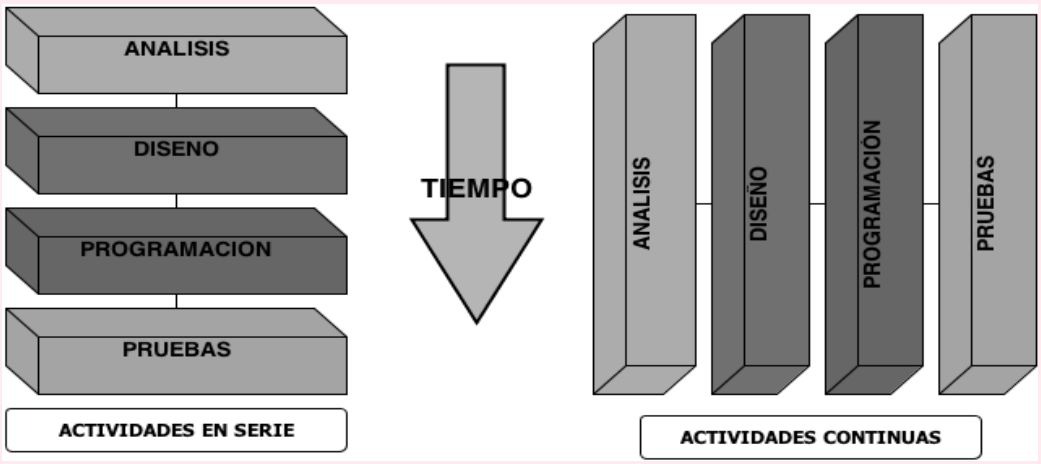
Escrito por: **Sergio Infante Montero** (Ingeniero de Software)

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**
http://about.me/neosergio
Twitter: **@neosergio**

En un típico proyecto ágil no hay roles definidos. Todos pueden hacer de todo y dentro de todo ese caos, confusión y la no existencia de una estructura jerárquica, los equipos producen normalmente software de alta calidad.

En el sentido tradicional en los equipos ágiles no existen los roles de analista, programador, evaluador, al menos no existen en el sentido tradicional y la razón fundamental de esta característica se debe a que las actividades de análisis, desarrollo, diseño y pruebas son actividades continuas a través de toda la duración del proyecto.



En palabras simples quiere decir que estas actividades no pueden estar separadas. Entonces, los equipos ágiles se caracterizan por los roles no tan definidos, por las actividades de desarrollo continuo y por el trabajo en equipo que producen siempre valor para el cliente.

Lo que hace funcionar al equipo

Hay ciertas cosas que se deben hacer para que un equipo funcione, estas son:

Involucrar a los clientes

A pesar de que en algunas ocasiones pueda apreciarse como innecesario o incluso como una perdida de tiempo, involucrar al cliente es sumamente importante. Es la mejor forma de entender las necesidades de ellos y poder llevar un proyecto al éxito.

Cuando el cliente está involucrado, la retroalimentación que brinda se convierte en el factor clave del éxito de un proyecto, las demos permiten ajustar las funcionalidades a la realidad, los clientes se convierten en parte importante del equipo.

Es por esta razón que metodologías como Scrum o XP constantemente promueven mediante sus prácticas que el cliente se involucre en el proceso

> *Principio ágil: Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.*

Sin embargo a pesar de que esto suena muy bien y parece lo más lógico, resulta que no siempre podemos involucrar al cliente y aquí es donde empiezan los tropiezos ¿qué hacer? Sin importar cuál es el motivo de la falta de compromiso, se necesita tener la credibilidad por parte del cliente. Una buena forma de hacerlo es tomar un problema y resolverlo en períodos quincenales, de forma tal que los clientes se darán cuenta y prestarán atención al trabajo que estás haciendo, demostrando que realmente puedes resolver problemas y no sólo planificar su resolución.

Pueden haber muchas razones por las cuales un cliente no se comprometa con el proyecto: demasiado trabajo, otros proyectos o quizás no necesitan el software urgentemente, pueden estar realmente muy ocupados; sin embargo debes recordar que para construir credibilidad y hacer que los clientes se comprometan, necesitas ganarte su confianza y eso, se logra cubriendo necesidades, solucionando problemas, actuando; prueba hacer eso y eventualmente te ganaras su compromiso.

Compartir el ambiente de trabajo

Algo que aumenta mucho la productividad del equipo es trabajar uno al lado del otro. Se trabaja mejor, las consultas se resuelven rápidamente, la interacción entre los

miembros del equipo mejora, la confianza entre los miembros de equipo se construye rápidamente.

Pero ¿qué pasa si no hay posibilidad de compartir un mismo entorno de trabajo, quizás porque no se encuentra en la misma ciudad o incluso en diferentes países? ¿Se pueden tener equipos ágiles distribuidos? Pues claro que sí. Sin embargo, los equipos que están en un mismo lugar siempre tendrán mayor ventaja.

Una recomendación en el caso de equipos distribuidos es que al inicio del proyecto, exista una semana para que los integrantes del equipo se conozcan, compartan tiempo tal vez jugando, haciéndose bromas, comiendo juntos... eso ayudará a establecer confianza entre los miembros; juntarlos físicamente durante una semana será de mucha ayuda. Luego de este tiempo se debe usar toda herramienta disponible de comunicación, como video conferencias diarias, herramientas de social media, mensajería, entre otros; de esta manera el equipo distribuido puede sentirse como un equipo que comparte un mismo espacio geográfico.

Auto organizados

El equipo ágil debe tener un objetivo en común, de manera tal que como equipo se tenga una meta clara. Para que esto funcione, el equipo necesita ser auto organizado.

La auto organización trata de alejar el ego y trabajar en conjunto para lograr el objetivo común, con todas las habilidades, talentos y pasión para entregar lo mejor en el proyecto.

Todos los miembros del equipo pueden ayudar en diversas tareas. Esto no significa que deban ser expertos en todo sino, se trata de reconocer cuál es la mejor manera de llevar a cabo el proyecto, haciendo que los roles dentro de los mismos encajen en los perfiles de los miembros y no que los miembros encajen en el rol.

Un equipo es auto organizado si:

- Crea el plan, hace las estimaciones y toma el liderazgo del proyecto.
- Se preocupa menos en los roles y cargos y está mas interesado en producir software que funciona.
- Toma iniciativas y no se sienta a esperar órdenes.

> *Principio ágil: Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.*

Multifuncional

Un equipo multifuncional es aquel que puede servir al cliente desde el principio hasta el fin. Tiene las habilidades necesarias y la experiencia para tomar cualquier requerimiento y convertirlo en una característica que funciona en un corto tiempo.

El secreto está en tener en el equipo personas que se sienten cómodas haciendo

diversas cosas, no solamente en sus áreas de confort.

Los especialistas son necesarios en ocasiones cuando el equipo se encuentra trabado en algún problema específico. Sin embargo, la mayoría de equipos se mantienen juntos y trabajan juntos como uno en toda la duración del proyecto.

Lo resaltante de los equipos multifuncionales es la velocidad a la que hacen el trabajo, ya que no tienen intervalos de pausa, esperando permisos o negociaciones de otros, pueden rápidamente entregar valor sin traba alguna.

Sólidos y confiables

Un buen equipo siempre confia en los resultados que produce; sabe que los clientes cuentan con ello y por lo tanto, no eluden la responsabilidad que conlleva el entregar algo de valor día a día.

Esta confianza se logra cuando el equipo es sólido, toma sus propias decisiones, hace lo que cree que es correcto para el cliente, resuelve sus propios problemas y no espera el permiso de nadie para hacer su trabajo. Siempre habrá ocasionalmente errores, pero vale la pena tomar el riesgo.

La confianza se va afianzando cuando el equipo empieza a tener contacto con las personas que confían en ellos, que vienen con problemas reales, que necesitan software para mejorar sus vidas. El hecho de tener demos constantes pone al equipo en el compromiso de mejorar entre presentación y presentación.

Principio ágil: Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan y confiarles la ejecución del trabajo.

Tengo que desarrollar un nuevo Software ¿por dónde empiezo?

Sin lugar a dudas, éste es el problema más frecuente al que nos enfrentamos los programadores: ¿por dónde comenzar a desarrollar una aplicación? Te enloqueces buscando consejos, tratando de elegir la metodología apropiada pero alguna vez ¿te preguntaste cuál es el método que mejor se ajusta a tus rasgos de personalidad? Este artículo abarca tanto aspectos técnicos y conceptuales como rasgos psicológicos relativos a cada individuo, que pueden ayudarte a tomar la decisión que mejor se ajuste a tu perfil.

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 tenemos que comenzar un nuevo proyecto de desarrollo de Software, dos sentimientos encontrados se nos presentan: el entusiasmo por empezar un proyecto nuevo y la sensación de tener miles de ideas pero sin embargo, ni una que nos indique cómo iniciar el camino.

Son sensaciones mucho más comunes de lo que se cree y se presentan hasta en los desarrolladores más experimentados. El tema es ¿cómo abordar el problema? Si aún continúas experimentando estas sensaciones es hora de que comiences a prestarte más atención a ti mismo y te plantees qué camino se adapta mejor a tu personalidad para así, lograr alcanzar el éxito que tanto deseas y muchas veces te deja con esa sensación de no haberlo alcanzado.

Si estás por comenzar un nuevo proyecto, presta atención a las siguientes páginas y espera llegar al final del artículo para asegurarte tomar la decisión más acertada.

1. Describe la visión global de la aplicación

Lo primero que uno debe tener muy en claro es **"de qué se trata la aplicación que se necesita desarrollar"**. Tenerlo en claro, no significa describir con precisión -siquiera aproximada- las características de la aplicación ni mucho menos, sus funcionalidades. Por el contrario, implica tener una visión global de lo que se necesita.

Pero **¿qué significa entonces, tener una visión global de la aplicación?** Puedes imaginar esa "visión global" como una breve descripción que te ayudará a "vender" tu aplicación. Alguna vez habrás visto un programa de televisión, donde el conductor o conductora de TV te invita a probar determinado producto. Es sabido que el tiempo en televisión "es tirano" así que el conductor, "lee" un rápido anuncio. Ese rápido anuncio, a veces es solo "un gancho" para generarte curiosidad. Pero la mayoría de las veces, te deja muy en claro de qué se trata el producto. Y eso mismo, es la visión global que tu debes tener con respecto a la aplicación.

La visión global, entonces, debe ser una breve descripción que en lo posible, no supere un párrafo y a lo sumo, se concluya en dos. Incluso, más ideal es aún, poder describir la aplicación en una única frase.

Una forma simple de intentar obtener una visión global de la aplicación, sería imaginar el anuncio que harías para ofrecerla. Te doy algunos ejemplos:

1. Con AppName podrá llevar un control preciso en tiempo real, de los productos disponibles en su stock y obtener alertas inmediatas cuando el inventario de existencias sea bajo.
2. Ingresa a AppName.com y encuentra el curso que estás buscando. Docentes especializados en diversas áreas, te esperan para que asistas a sus clases en línea.
3. Encuentra amigos o citas en SiteName.com. Muchas personas esperan conocerte. Dinos cómo es la persona que deseas encontrar y te presentaremos a quienes se ajusten a tu perfil.

Diferentes plataformas, diferentes objetivos, diferente público. Sin embargo, no cuesta trabajo imaginarse de qué se trata cada una de las aplicaciones.

Una vez tienes "tu anuncio", convertirlo en una descripción menos personal, no es difícil. Solo se trata de redactarlo de manera impersonalizada y tratando de reducir al máximo posible, los adjetivos que generan indirectamente, una opinión previa del producto:

1. Sistema de control de *stock* en tiempo real y alertas inmediatas de inventario.
2. Plataforma *online* de búsqueda y contratación de cursos a distancia sobre diversas áreas de conocimiento.
3. Plataforma de citas *online* con sistema inteligente de contraste de perfiles y

examen de compatibilidades.

A partir de tu visión global, te será mucho más simple entender qué es aquello que necesitas para alcanzar tus objetivos.

2. Describe las particularidades de tu aplicación, solo cuando te sientas seguro de poder hacerlo

Es muy importante poder entrar en cuestiones más particulares sobre la aplicación, antes de decidir cómo vas a desarrollarla y qué tecnologías implementarás.

Pero para entrar en cuestiones particulares, debes evitar los detalles y solo concentrarte en aquellas cosas que la aplicación podrá hacer, es decir, en la descripción general de sus funcionalidades.

Definir las generalidades de las funciones de un sistema, no es poca cosa. Es aquí donde los problemas de organización y análisis de requerimientos comienzan a salir a la luz.

Existen dos métodos probados que puedes emplear para esto, pero ninguno de ellos te asegura resultados óptimos. Por ello, **antes de decidir qué método deseas emplear, es muy importante que te tomes el tiempo de "definirte a ti mismo"**, ya que el resultado de cada método, **dependerá de tus fortalezas y debilidades**.

> *Nunca olvides que conociendo tus debilidades, te abres camino para poder trabajarlas y convertirlas en fortalezas*

Los métodos que puedes emplear, son:

1. **Método tradicional:** consiste en hacer una lista de funcionalidades conocidas de la aplicación. Por ejemplo:
    - ABM de productos y categorías
    - Generación automática de reportes de *Stock*
    - Generación automática de alertas cuando el *stock* llegue a *N* productos

    Este método es el más común, pero sin embargo, te enfrenta a ciertos desafíos difíciles de resolver a la hora de pasar a los detalles. Este método, **puede ser contraproducente en los siguientes casos:**

- Eres una persona que se distrae con facilidad;
- Generalmente, te cuesta organizarte;
- Sueles concentrarte mucho en los detalles y con frecuencia, pierdes el objetivo original de vista;
- Te sueles "enredar" a menudo cuando intentas planificar lo que debes hacer para cumplir un objetivo.

Por el contrario, el mismo método **puede ser productivo cuando:**

- Te resulta extremadamente imposible (o directamente imposible) convivir con incertidumbres (no saber o desconocer lo que sucederá);
- A menudo, sientes la necesidad de mantener el control sobre situaciones y/o personas que te rodean;
- Con frecuencia, puedes anticiparte a las consecuencias resultantes de tus decisiones;
- Frente a diferentes circunstancias sueles experimentar una especie de bloqueo frente a lo desconocido;
- Generalmente, tienes la sensación de no saber si eres o no capaz de lograr aquello que te propones.

2. **Método ágil:** consiste en definir las acciones que hará el sistema, desde la óptica de quiénes lo utilizarán. Para ello, se utiliza un formato descriptivo llamado "Historia de Usuario". Una historia de usuario se arma con la frase: *como [mi rol de usuario] puedo [acción]*. Opcionalmente, puede completarse con: *para [beneficio]*. Por ejemplo:
    - Como administrador del sistema puedo agregar categorías
    - Como administrador del sistema puedo editar categorías
    - Como administrador del sistema puedo eliminar categorías preexistentes
    - Como responsable de depósito puedo ingresar un producto al inventario
    - Como responsable de depósito puedo obtener un reporte del stock de los productos existentes en el depósito
    - Como vendedor puedo consultar el *stock* de un determinado producto

Este método, es mucho menos utilizado que el tradicional. Sin embargo, estadísticamente ha demostrado arrojar mejores resultados en el software final generado. No obstante, **puede presentarse como un obstáculo, cuando:**

- A menudo y en diferentes situaciones experimentas sensaciones de

ansiedad sin origen aparente;

- Sueles experimentar con mucha frecuencia una sensación de bloqueo, estancamiento, avance lento o crisis de ansiedad frente a circunstancias no previstas;

A diferencia del anterior método, este otro puede resultar **sumamente productivo** una vez se supera (o se aprende a manejar) el miedo a lo desconocido, sin mayores mayores requisitos. No obstante, ciertas características a tener en cuenta al momento de elegir esta opción, son:

- Eres una persona segura de sí misma y con frecuencia sientes que puedes alcanzar los objetivos que te propones;
- Te estresas con facilidad frente a las grandes lluvias de ideas o grandes cantidades de información y te sientes mucho más cómodo y distendido, haciendo las cosas "paso a paso";
- Tienes facilidad para manejar las sensaciones que te generan los cambios de planes o te adaptas con facilidad a los cambios sin que ello implique no poder continuar con tu vida diaria de forma natural.

> *Elegir aquel método que mejor se adapte a tu personalidad, evitando ser tu quien deba adaptarse al método, será la principal fuente de garantía sobre los resultados que finalmente obtengas.*

3. Prioriza y valora las funcionalidades del sistema

En principio, debes tener en cuenta que hayas optado por un método tradicional o uno ágil, las funcionalidades del sistema pueden ir cambiando y/o ampliándose con frecuencia. No obstante, suelen tenerse mayor cantidad de funcionalidades descritas cuando se trabaja de forma tradicional que de forma ágil. Sin embargo, un buen número de funcionalidades descritas para poder comenzar con el desarrollo, suele ser inferior o igual a 10. Si puedes "frenar" un poco en ese punto, te será mucho más fácil avanzar en el desarrollo. Es en este punto donde puedes comenzar a priorizar y valorar las funcionalidades.

Prioridad

La prioridad de una funcionalidad, es una cuestión de dependencia lógica. La mejor estrategia para darle prioridad a una funcionalidad sobre otra, es preguntarse ¿puedo contar con la funcionalidad "X" sin depender de la funcionalidad "Y"? Cuando no sabes qué responder a esa pregunta, puedes llegar un poco más a fondo preguntándote:

¿depende “X” de alguna otra funcionalidad?

- Cuando “X” dependa de “Y”, “Y” tendrá prioridad sobre “X”.
- Cuando “Y” dependa de “X”, “X” tendrá prioridad sobre “Y”.
- Cuando “X” no dependa de “Y” e “Y” no dependa de “X”, podrá tener mayor prioridad, aquella que mayor valor tenga para la aplicación.

Valor

El valor que una funcionalidad tenga para la aplicación, a diferencia de la prioridad, no es una cuestión de dependencia, sino de relevancia para el negocio. *¿Qué funcionalidad generaría más éxito a mi aplicación?* Se puede decir que el valor que uno le otorgue a una funcionalidad, será relativo a las consecuencias que dicha funcionalidad generará para el negocio. Por ejemplo, en un sistema de control de inventario, se cree que generará un mayor éxito a la aplicación la posibilidad de generar reportes de *stock* que la de administrar categorías de productos.

Prioridad y valor: ¿cómo otorgarlos?

Al momento de dar prioridad y valor a un pequeño número de funcionalidades, deberás tener en cuenta lo siguiente:

- La prioridad es única para cada funcionalidad y se aplica de forma incremental. A menor número, mayor prioridad. Es decir que la funcionalidad con prioridad 1 será prioritaria frente a la de prioridad 2.
- La valoración se mide en base a una escala simbólica que puede ir de 1 a 100 en incrementos de 10 o 5. A mayor número, mayor será la relevancia de esa funcionalidad para el sistema. Pueden coexistir múltiples funcionalidades con la misma relevancia.

4. Detalla la funcionalidad con mayor prioridad

Una vez hayas priorizado y valorado al menos un total de entre 5 y 10 funcionalidades, puedes tomar aquella a la cuál le hayas asignado la mayor prioridad y comenzar a definir sus **criterios de aceptación**.

Los criterios de aceptación de una funcionalidad, abarcan hasta el más mínimo detalle sobre ésta. Los mismos, van desde aquello que verá el usuario al ingresar a dicha funcionalidad hasta los pasos que deberá seguir para concretar el proceso y la forma en la que el sistema deberá responder y reaccionar frente a dichas acciones del usuario.

Por ejemplo:

Tomemos el caso de la funcionalidad que tradicionalmente la hemos definido como `Generación automática de reportes de Stock` y ágilmente dijimos que `Como`

responsable de depósito puedo obtener un reporte del stock de productos.

Los criterios de aceptación para esta funcionalidad podrían describirse detalladamente como los siguientes:

- Desde la pantalla principal del sistema, el usuario deberá pulsar un botón con la leyenda “Generar reporte de Stock”.
    - El sistema deberá mostrar en la siguiente pantalla, una lista con los nombres de los productos, el código de barras, el sector del depósito en el cual se ubica cada producto (letra y número de sector) y la cantidad de existencias.
        - Cuando la existencia sea inferior o igual a 50, se deberá mostrar el stock en rojo.
        - Cuando la existencia sea menor a 25, se deberá mostrar un botón a la derecha del stock, con la leyenda “Alertar al responsable de depósito”.
            - Al pulsar dicho botón, se enviará un e-mail a la cuenta del responsable de depósito (responsable@example.org), con asunto “Existencia insuficiente del producto {código de barras del producto} (stock: {stock})”.
            - El e-mail no podrá enviarse con una frecuencia inferior a 24 horas.
            - Cuando dicha existencia insuficiente haya sido reportada, el sistema deberá quedar al pendiente y transcurridas 72 horas desde el envío de la primera alerta, deberá constatar si se han repuesto existencias. De no haberse repuesto, deberá enviar el mismo e-mail de forma automática, al responsable de depósito y al Gerente de Producción (gerente@example.org).

5. Define las tareas necesarias para que cada funcionalidad sea desarrollada

Cuando los criterios de aceptación de al menos 1 funcionalidad se encuentren definidos y 5 o 10 funcionalidades hayan sido descritas, estarás en condiciones de evaluar qué tecnologías y/o herramientas serán convenientes para el desarrollo de la aplicación. Para ello deberás “colocar en la balanza”:

- La visión global de la aplicación;
- El conjunto de funcionalidades ya descritas;
- La base de referencia de los criterios de aceptación especificados;
- Los recursos de los cuáles dispones;
- Tu nivel de conocimientos;
- El tiempo que puedes invertir en investigar antes de tomar la decisión tecnológica apropiada; y...
- Factores ajenos a tu responsabilidad o alcance, que impacten de forma directa sobre la aplicación.

Haciendo un balance de los 7 puntos anteriores, podrás tomar la decisión tecnológica más apropiada y cuando lo hagas, estarás en condiciones de comenzar a definir las tareas para iniciar el proyecto y las requeridas por cada una de las funcionalidades.

Las **tareas iniciales del proyecto**, suelen basarse en todo aquello que debes hacer para preparar tu entorno de desarrollo y posiblemente, el núcleo o motor bajo el cuál funcionará la aplicación. Estas tareas, suelen hacerse al inicio del proyecto y luego, en eventuales ocasiones, requerirán algún tipo de adaptación pero con poca frecuencia.

Las **tareas específicas a cada funcionalidad** son tal vez, las que mayor "ruido" generan al momento de plantearse "¿qué debo hacer para lograr desarrollar esta funcionalidad?".

Básicamente, deberás escribir una lista (tal cuál vayan apareciendo las ideas), de cada una de las cosas que necesitas hacer para que dicha funcionalidad pueda ser desarrollada. Estas tareas pueden ser de diseño gráfico, maquetación, programación, arquitectura, DBA, *testing*, QA hasta incluso, tareas administrativas.

Hacer una lista independiente de las tareas necesarias para desarrollar cada una de las funcionalidades de la aplicación, te permitirá:

- **Discriminar y agrupar tareas por tipo de tarea:** por ejemplo, las de diseño gráfico, las de programación, las de investigación, las de documentación, etc.;
- **Organizar el orden en el que vas a llevarlas a cabo:** de la lista de tareas y a simple vista, podrás ver aquellas que necesites emprender antes que otras. No será difícil notarlo: bastará con que te preguntes *¿puedo hacer la tarea X antes que la tarea Y?* Si una tarea no depende de otra, la puedes desarrollar en el momento que desees. Guíate por el método que utilizaste al momento de priorizar funcionalidades y aplícalo a las tareas.

¿Cómo seguir?

Al **trabajar de manera tradicional**, los pasos 2, 3, 4 y 5 suelen realizarse **de forma consecutiva hasta alcanzar la última funcionalidad** y el último detalle requerido por ésta. Los mismos, son plasmados en un documento denominado "Documento de Alcance" y no se comienza con el diseño y desarrollo de la aplicación, hasta que dicho documento se encuentra finalizado y aprobado por los principales interesados en el proyecto.

Cuando **se trabaja de forma ágil**, el paso 2 se realiza en forma paralela a los siguientes e incluso, mientras que se está desarrollando la aplicación, al igual que sucede con el paso 3. Los pasos 4 y 5, se realizan cíclicamente. Al comienzo del proyecto, se establece la duración de un ciclo de desarrollo (por ejemplo, cada 2 semanas se decide cumplir un ciclo). Dependiendo de este tiempo, se eligen una, dos (o más) historias de usuario, se efectúan los pasos 4 y 5 al comienzo del ciclo, se diseñan y desarrollan las historias de usuario y luego, se repite el ciclo desde el paso 4.

FirefoxOS App Days Barcelona

El pasado 26 de enero de 2013 se celebraron los FirefoxOS App Days alrededor de diversas ciudades del mundo. Hackers & Developers Magazine fue invitada para cubrir el evento y Laura Mora, estuvo en Barcelona. En este artículo, nos cuenta cómo fue este significativo día y nos regala una entrevista con Alina Mierlus, una de las organizadoras.

Escrito por: **Laura Mora** (Administradora de Redes y Sistemas GNU/Linux)

Laura es administradora de **Redes y Sistemas GNU/Linux** en Cataluña. Tiene una larga trayectoria en la consultoría de soluciones telemáticas **open source** para movimientos sociales. La base de sus proyectos es el **empoderamiento tecnológico** de las personas.

Webs:
Blog: http://blackhold.nusepas.com/
Web: http://delanit.net

Redes sociales:
Twitter / Identi.ca: **@Blackhold_**

El pasado 26 de Enero se celebró en el espacio MOB (Makers of Barcelona) en la calle Bailén de Barcelona, el Firefox APP Days, un evento de Mozilla para presentar el nuevo sistema operativo para dispositivos móviles, FirefoxOS. Este evento se llevó a cabo en la misma fecha en otras ciudades del mundo, no solo para dar a conocer este nuevo proyecto, sino también, para que los programadores se animen a sumarse en el desarrollo de aplicaciones para esta plataforma.

El encuentro tuvo un formato participativo y lúdico, lo que hizo que fuese uno de aquellos días curiosos e inolvidables.

La jornada empezó con una rueda de presentaciones de los distintos asistentes, la gran mayoría desarrolladores, pero entre ellos también administradores de sistemas, *Community Managers* y algunos periodistas.

Se siguió con una breve actividad en la cual los organizadores formulaban unas afirmaciones y los asistentes teníamos que posicionarnos en la sala según nuestra aceptación o rechazo. Algunas de ellas hacían referencia a aplicaciones de mensajería en dispositivos móviles o si una aplicación podría sustituir a nuestro médico de cabecera.

Más tarde, por grupos nos pusimos a hacer un tormenta de ideas sobre aplicaciones que podrían ser interesantes. Tras ésta, se mostraron las ideas de los distintos grupos y los participantes marcaron con un punto las aplicaciones que les resultaban de mayor interés.

Al finalizar la mañana, con los mismos grupos que habíamos hecho la tormenta de ideas fuimos pasando por distintas mesas donde un experto de Mozilla nos explicaba conceptos o programas necesarios para entender la web y el desarrollo de las aplicaciones Web para FirefoxOS. Algunas de estas charlas livianas, hicieron referencia a *Responsive Web Design*, Node.Js, APIs *client side storage*, plantillas para las F*irefoxOS Web Apps*, etc.

Por la tarde los asistentes se pusieron a conocer distintas partes del sistema operativo, desde el desarrollo de aplicaciones, funcionalidades del sistema, instalación en dispositivos, etc.

Desde Hackers & Developers Magazine, tuvimos la posibilidad de entrevistar a una de las piezas que hizo posible esta FirefoxOS APPs days en Barcelona, Alina Mierlus y esto, fue lo que nos dijo:

Alina Mierlus, ingresó en la comunidad del Open Source hace poco más de 9 años y desde hace 4, contribuye a Mozilla organizando y facilitando eventos de la comunidad.

¿Cómo empezó la idea de gestarse un sistema operativo para dispositivos móviles desde la comunidad de Mozilla?

Una cosa muy bonita dentro de Mozilla es que la gente tiene la libertad de pensar, proponer sus ideas y sus proyectos. Todo empezó entre cuatro desarrolladores de Estados Unidos en una lista de desarrollo. Planteaban el cómo se podía ir más allá del límite actual del web ¿qué se podría hacer? y pensaron precisamente en esto, en un sistema operativo para móviles. Uno de los motivos es que estos dispositivos son los únicos [para] los cuales no se había desarrollado aún la integración con el web, tal como tenemos en el escritorio.

***«Boot to Gecko»* era el nombre inicial de este sistema operativo ¿por qué la comunidad se decantó inicialmente por este nombre?**

Prácticamente porque el motor, el *engine* integrado en Firefox, se llama Gecko y boot to gecko, significa *"boot to the web"*.

¿Cómo enfocáis el tema de la privacidad de los datos?

Creo que son dos cosas un poco distintas. Para el tema privacidad, Mozilla tiene una serie de productos, por ejemplo, persona. Persona es un sistema de identidad para la web, es prácticamente como el OpenID pero todo es web. Esto significa que es aún más libre, porqué sólo necesitas conectarte al web para hacer login.

Persona funciona como un servicio. Ahora mismo tenemos un montón de problemas, por ejemplo, tenemos contraseñas en diez, veinte, web sites, implementando persona, lo que podemos hacer es resolver esta problemática. Persona es uno de los productos que intenta a ayudar a la gente con los temas de privacidad.

Después, hay programas como el *webmaker* que enseña a la gente como es el web y cómo funciona.

Después en firefoxOS si miráis el tema de *settings*, está la opción *"do not track"* (no quiero ser rastreado). Esto dice a las *webs* y otras aplicaciones que no quiero que coja datos sin mi permiso.

Pues con persona, que es el sistema de identidad basado en el web, es una API abierta que cualquier desarrollador puede usarla y hacer *deployment* en su aplicación.

¿y el tema de la intercomunicación de la terminal con otros dispositivos?

Prácticamente todo se hace desde el web

¿Sería como subir cosas en el cloud?

Exacto. La base de FirefoxOS son las *webapps* y toda la comunicación se hace mediante el web.

Entonces ¿qué compatibilidad tenemos con aplicaciones de otros fabricantes?

En FirefoxOS sólo tenemos aplicaciones web y ahora tenemos que realizar un gran trabajo en convencer a los desarrolladores a crear aplicaciones basadas en el web para que sean compatibles con FirefoxOS. Es un mercado un poco complejo (...) y esto no nos gusta porque

es un dolor de cabeza para el desarrollador. Esto de la intercomunicación es una tarea que depende de nosotros, de la comunidad y de la gente que desarrolla apps.

¿Java no hay, no?

Exacto. Está JavaScript, que tu puedes hacer una app para FirefoxOS y ya te funciona en el navegador de GNU/Linux y de cualquier otro sistema operativo (...) y este es el reto: de que mas y mas personas se den cuenta de esta oportunidad de construir *webapps* y liberar así las aplicaciones a más y más usuarios.

¿qué aceptación ha habido por parte de los fabricantes de dispositivos móviles de esta iniciativa?

Hay ya algo de información sobre esto. Los primeros terminales para desarrolladores vendrán de *Geeksphone* y el primer terminal con el que se han hecho pruebas con FirefoxOS ha sido el ZTE, de una empresa china. A nivel comercial no tengo mas información.

¿y por parte de las operadoras? Hemos visto que aquí en el encuentro ha habido algunos desarrolladores de Telefónica. ¿Qué aceptación ha habido? porque las operadoras son las que acaban decidiendo cuáles son los dispositivos que venden a sus usuarios.

El papel de las operadoras es muy importante. Si eres un *geek* y tienes conocimientos puedes descargar el código de GitHub, hacer un *build* y ponerlo en tu teléfono, pero claro, la idea es llegar al máximo numero de usuarios posible.

De momento Telefónica ha dado un gran soporte a FirefoxOS, sobretodo para empezar a construir el producto de forma conjunta.

También hay otras operadoras que lo redistribuirán, tal como Deutsche Telecom, Telefónica en Sudamérica y en España, etc. Estas serán las encargadas de acercar este sistema operativo a los usuarios, pudiendo modificar también la parte superior, el Gaia.

¿Alguna previsión de fechas en la que podamos ver FirefoxOS en las tiendas?

En España no hay ninguna previsión, pero hay previsión para este mismo año poder verlos en Brasil.

En la página web de FirefoxOS, podréis ver un listado de operadoras y fabricantes que le han dado soporte, pero más allá de qué, cuando, quien y cómo, aún no se sabe.

Al ser un sistema libre ¿cómo veis el tema de la modificación por parte de los operadores y fabricantes de este sistema operativo?

Esta es una pregunta interesante, la vemos que se plantea muchas veces dentro de la comunidad. Por ejemplo, en la comunidad Catalana siempre hemos tenido Firefox en Catalán y esto es porqué lo hemos decidido así. Con FirefoxOS la cosa cambia un poco, porque para tener éxito es necesario [ser *partners* de] las operadoras. Es normal, porque es como llegas al mercado.

Personalmente soy bastante positiva, porque lo veo que es una relación que se irá formando de cara al futuro. No creo que debamos ser negativos.

Las operadoras sólo tendrán la capa de arriba y podrán añadir cosas, pero ya lo iremos viendo.

¿Se ha probado firefoxOS en openmoko? ¿qué tal la compatibilidad? (openmoko es hardware libre)

Por lo que sé, no. La única cosa que conozco es que se ha hecho un *port* para rasberri Pi, este ordenador pequeño. Pero con openmoko, no lo he oído aún. Se puede hacer, porque FirefoxOS es un sistema libre y es posible hacerlo.

¿Y tenéis también un emulador de FirefoxOS que lo podemos probar en nuestro Ordenador?

Si. Más que un emulador, es una cosa más sencilla, un simulador: FirefoxOS Simulator,

que se puede instalar como una extensión de Firefox.

¿Porqué FirefoxOS? ¿alguna aportación para animar a la gente a pasarse a éste sistema operativo?

Yo creo que es una gran oportunidad, no sólo para los usuarios, sino para todas aquellas personas que empiezan a desarrollar aplicaciones para dispositivos móviles, porque el entorno de desarrollo es muy fácil, porque es prácticamente la tecnología web nativa, HTML5, JavaScript y CSS. Además al ser código abierto puedes modificarlo prácticamente todo, permitiendo a la gente trabajar a distintos niveles y modificar las aplicaciones para sus necesidades. ¿Porqué firefoxOS? pues porque queremos probar el límite del web.

¡El límite lo pone la imaginación!

¡Exacto! Imagínate que en pocos meses, *webapps* pudiesen solucionar problemas reales de la sociedad, por ejemplo una *webapp* desarrollada en colaboración con una asociación de médicos de tu población...

Ya lo comentabais en la dinámica de esta mañana ¿hasta qué punto consideráis que una aplicación web podría reemplazar vuestro médico de cabecera?

¡Exacto! son preguntas que nos tenemos que hacer a veces y venir con cosas innovadoras como desarrolladores, porqué se están cambiando muchas cosas. En Europa se apuesta mucho por la creación de innovación. FirefoxOS quiere abrir el ecosistema para todas estas cosas, al mismo tiempo que dar la oportunidad a desarrolladores de interaccionar directamente con el usuario, por ejemplo a través del *marketplace*, que es un openAPI. Todo en FirefoxOS es webAPI.

¿Todas las aplicaciones del *marketplace* (webAPI), podrás usarla y modificarla?

¡Exacto! Tu coges las cosas y las pones en tu servidor y puedes construir en torno de esta webAPI. Por ejemplo el *marketplace* de Mozilla es un webAPI que lo puedes coger y puedes hacer tu propio *marketplace* y poner ahí la aplicación que tu quieras. No hay estas restricciones, ni tampoco la de tengo que pagar una cuota mensual por poner mi aplicación ahí.

Pues muchas gracias y ya con esta explicación, si no os pasáis a FirefoxOS es porque ¡os lo estáis perdiendo! ¡Muchas gracias y hasta la próxima!

Gracias!

Bambi

por **Susie Oviatt**

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.

Mensajes de Nuestros Lectores

Luis Felipe Dominguez Vega (Cuba)
Buenas, ante todo felicitar al equipo de desarrollo de esta grandiosa revista, de la cual soy un asiduo lector y seguidor. Soy un estudiantes en Ingeniería en Ciencias Informáticas de la Universidad de las Ciencias Informáticas en Cuba, a través de un sitio creado por nuestra comunidad (humanOS) puedo descargar las actualizaciones de su revista y por tanto nutrirme de tan excelentes artículos elaborados por tan buenos desarrolladores. Les comento que soy desarrollador en C++ y soy de los que utiliza solamente Linux (cualquier distribución, a cada rato cambio, pero siempre Debian será la casa a donde vuelva siempre). Les comento que gracias a los posts de Eugenia Bahit me he incursionado en el desarrollo de aplicaciones Web y sobre todo en el Patrón Arquitectónico MVC utilizando siempre FrontController, mis primeros pasos los di leyendo sus artículos. Por lo general me encantan los conocimientos que adquiero leyendola. Sigan así y que la comunidad continúe creciendo, tienen un fiel seguidor por aquí.

Eduardo Arcentales A. (Ecuador)
Acabo de descubrir su revista, me está gustando mucho y espero que puedan incluir más temas sobre agilismo (…)

Reynaldo (México)
Buen día equipo de HDMagazine me llamo Reynaldo del estado de Yucatán, México, déjenme decirles que les sigo desde que salio la primera revista electrónica y por cierto estoy encantado con la información que ustedes nos ofrecen. (Mi duda), se que conocen muy amplio el mundo del software libre y me gustaría saber cual lenguaje de script utilizar para iniciar en desarrollo web ademas de PHP. ¿Cual me sugieren aprender como primera opción (Python o JavaScript)? Tanto python como javascript son lenguajes interpretados y para web (sin olvidar a perl), pero tengo duda si python puede hacer lo mismo que javascript, aunque se que el primero puede usarse igualmente para aplicaciones de escritorio. Les mando un saludo :)

Respuesta: *Python y JavaScript son dos cosas totalmente diferentes. Mientras que JavaScript es un lenguaje que se procesa del lado del cliente, Python, lo hace del lado del servidor. Por otra parte, JavaScript está enfocado a aportar valor a aplicaciones Web, mientras que Python, es un lenguaje de más bajo nivel (aunque no permite interactuar con el Hardware de forma directa) con el que se pueden desarrollar aplicaciones no solo Web sino también de escritorio.*

¡GRACIAS POR LEERNOS!