5 Disciplina - un elemento importante

Revista de Software Libre ATIX 2008

Equipo Editorial

Colaboradores

Dirección y Coordinación General

Esteban Saavedra López (jesaavedra@opentelematics.org)

Diseño y Maquetación

Jenny Saavedra López (jennysaavedra@hotmail.com)

Esteban Saavedra López (jesaavedra@opentelematics.org)

Herramientas

La edición de esta revista fue realizada de forma integra haciendo uso de Software Libre

Palabra quechua,
con un sentimiento profundo
y con gran significado filosófico

El que lo sabe

El que lo intenta

El que lo puede

El que lo logra

El que lo sabe, el que lo intenta, el que lo puede, el que lo logra

Editorial

Todos recordamos, las palabras de nuestros padres cuando eramos pequeños, donde nos recomendaban qué debíamos hacer y qué no; en realidad todas esas palabras nos fueron ayudando a comprender las normas y las formas de organizar todas y cada una de nuestras actividades. Hoy en día estamos seguros que esas palabras fueron las que nos impulsaron a mantener la disciplina en todo lo que hacemos.

Si nos detenemos a pensar y meditar la realidad del mundo actual, observaremos realmente cuanto ayudó y ayuda la disciplina en el mundo, a nivel de países, de empresas, y de personas. La disciplina se convierte en un elemento esencial para lograr progreso, bienestar y supervivencia; es por eso que en el tema de proyectos de software libre, la disciplina se convierte en un elemento indispensable, que al ser aplicado de forma correcta, permitirá que nuestros proyectos prosperen y sean exitosos

Disciplina – Un elemento importante, un título que contiene de por si, un elemento que debe estar presente en todo proyecto y en toda actividad de nuestras vidas, de tal forma que al ponerla en práctica, lograremos tener mayor eficiencia y eficacia en la realización de los mismos, y nos permitirá cumplir las metas trazadas.

En éste quinto número ponderamos aspectos como la presencia de nuevos autores y la presencia de nuevas áreas de publicación, también empezamos con la participación de entusiastas del software libre de países amigos en América Latina, caso específico Venezuela; así mismo adelantarles que en los próximos números presentaremos notables sorpresas de interés para la comunidad de software libre de América Latina.

Mantener la disciplina, un paso hacia el progreso.

Bienvenidos a nuestro quinto número

Esteban Saavedra López
Director y Coordinador General

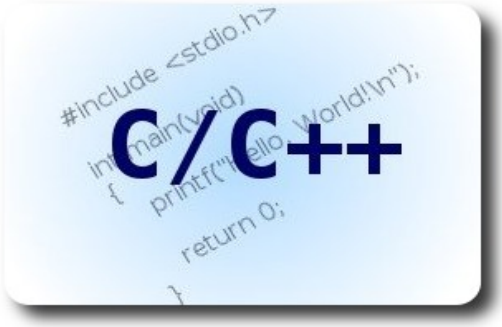
Contenido

Liberado el 10 de noviembre de 2008

5 Disciplina - un elemento importante

Utilizando herramientas de desarrollo C++ en Linux (1ra parte)

Desarrollar y construir sistemas en entornos *NIX puede llegar a parecer algo dificultoso al inicio, sin embargo, con el uso adecuado de las herramientas que proveen los entornos GNU, ésta se vuelve una tarea sencilla.



Introducción

Como desarrolladores, debemos conocer todos los estándares que exige el entorno. Este pequeño tutorial describe algunas prácticas base, cuyo objetivo no es el de profundizar a desarrolladores expertos en el tema, sino, el de dar a conocer las herramientas que pueden ser utilizadas de una forma bastante simple en cualquier shell, especialmente para los amantes de C.

Un entorno de desarrollo simple

Vamos a utilizar el lenguaje C++ para crear la parte lógica del desarrollo, es decir, para crear los programas.

Primero debemos verificar si existe un compilador GNU para C++ (debe quedar claro que no es lo mismo un compilador de C). Para comprobarlo debemos utilizar la siguiente orden en un shell:

```
arn@localhost:~$ g++
```

La salida debe verse:

```
g++: no input files
```

En caso de obtener la salida anterior, existe un compilador C++ instalado en el sistema. En caso de que la salida sea:

```
bash: g++: command not found
```

Se debe instalar el paquete build-essential de la siguiente forma (utilizar superusuario):

```
arn@localhost:~$ apt-get install build-essential
```

Nota: El paquete build-essential contiene todas las herramientas necesarias que se tocarán en este tutorial. Si su sistema no soporta el comando apt-get (que es típico de distribuciones tipo Debian), el paquete existe en todas las distribuciones: Ej: buscar paquetes RPM.

Ahora crearemos el código fuente de un "*hola mundo*". Con su editor de texto favorito crear el archivo

holamundo.cpp

```
#include <iostream>
int main(int argc, char **argv){
            std::cout << "Hola mundo" <<
std::cout;
            return 0;
}
```

Guardarlo y compilarlo utilizando la siguiente

orden en el shell:

```
arn@localhost:~$ g++ holamundo.cpp
```

Como resultado de esta compilación se creará el archivo a.out dentro del mismo directorio de trabajo. Notarán que este archivo tiene permisos de ejecución. Podemos comprobarlo listando sus atributos con:

```
arn@localhost:~$ ls -l a.out
```

```
-rwxr-xr-x 1 arn arn 8482 2008-10-23
11:47 a.out
```

Las X's significan permisos de ejecución. De éste modo ejecutemos el archivo generado con :

```
arn@localhost:~$ ./a.out
Hola mundo
```

Excelente!, nuestro primer programa en un entorno *NIX ha sido creado con éxito. Sin embargo, en un entorno de desarrollo más profesional, se necesita que las herramientas sean lo más versátiles posible. El compilador GNU de C++ utilizado tiene muchas más opciones, por ejemplo, para producir un ejecutable que tenga un nombre primerprograma, se debe compilar de la siguiente forma:

```
arn@localhost:~$ g++ holamundo.cpp -o
primerprograma
```

Y para ejecutarlo:

```
arn@localhost:~$ ./primerprograma
Hola mundo
```

Un proyecto de varios archivos

Hacer proyectos grandes cuyo código fuente conste únicamente de un archivo, es simplemente fantasioso o tal vez muy atrevido, pero por sobre todas las cosas, es muy poco práctico. Los programas siempre utilizan varios archivos fuente para poder compilarse en un solo ejecutable. En el siguiente ejemplo utilizaremos dos módulos:

- **`main.cpp`** donde reside el punto de inicio del programa(función main)
- **`libreria.h`** donde existe algunas rutinas de librería bastante básicas (funciones suma y resta)

`main.cpp`

```
#include <iostream>
#include "libreria.h"
int main(int argc, char **argv){
  int resultado = suma(4,6);
  std::cout << "El resultado de la suma
es: "
  << resultado << std::endl;
  return 0;
 }
```

`libreria.h`

```
int suma(int a , int b ){
    return (a+b);
}
int resta(int a, int b){
     return (a-b);
}
```

Para compilar este programa, se debe ejecutar:

```
arn@localhost:~$ g++ main.cpp libreria.h
-o programa
```

Estandarizando las librerías: archivos de cabecera

Las librerías de C y C++ en entornos de desarrollo *nix, se descomponen en

dos partes:

- **Declaraciones**: Archivos header que son una especie de "índice" de rutinas y demás utilitarios en las librerías
- **Las definiciones**: Archivos donde existe la implementación completa de las librerías

Para el ejemplo anterior, si estandarizamos la librería debemos dividirlo en dos partes **`libreria.h`** y **`libreria.c`**

libreria.h

```
#ifndef LIBRERIA_H
#define LIBRERIA_H
int suma(int, int);
int resta(int, int);
#endif
```

libreria.cpp

```
#include "libreria.h"
int suma(int a, int b){
    return (a+b);
}
int resta(int a, int b){
     return (a-b);
}
```

Debemos notar las definiciones **#ifndef LIBRERIA_H** , **#define LIBRERIA_H** y **#endif**, que son macros que evitan que el archivo sea incluido dos veces en un ejecutable (que sería un error de sintaxis). Es muy frecuente encontrar dichos macros en todos los archivos de cabeceras de sistemas en desarrollo.

La división de éste tipo usualmente por razones prácticas. En el archivo de cabecera .h se pueden encontrar todas las definiciones de rutinas, clases, enumeraciones, constantes y otras macros, de este modo, un desarrollador puede encontrar fácilmente algo que necesite, puesto que sirve como una especie de índice o resumen.

En este punto tenemos dos opciones:

- ✔ Compilar todos los archivos fuente (**main.cpp**, **libreria.h**, **libreria.cpp**) en un solo paso para crear el ejecutable.
- ✔ Compilar la librería independientemente(**libreria.h** y **libreria.cpp** del programa, cuyo código fuente luego se enlazará en un siguiente paso con la librería.

Para la primera opción procedemos de la siguiente manera:

```
arn@localhost:~$ g++ libreria.cpp
libreria.h main.cpp -o programa
```

La segunda opción es la más común, y es una forma más estandarizada de hacerlo: creando una librería independiente, para que en un paso posterior, hacer un enlace de dicha librería y el programa principal. Primero hacemos la creación de la librería

```
arn@localhost:~$ g++ libreria.cpp
libreria.h -c -o libreria.o
```

Este comando crea el archivo binario librería

La opción -c indica que solo se creará un archivo no ejecutable, puesto que **libreria.o** debe ser enlazado a otro código para ser ejecutado. Cabe destacar que no es necesario incluir el archivo de cabecera en el comando de compilación, puesto que dentro de **libreria.cpp** ya lo incluye, solo debe estar accesible. En nuestro caso, todos los archivos están en el mismo directorio, así que no existe problemas en éste aspecto, sin embargo, proyectos más grandes y estructurados necesitan otras directivas que se mencionaran más adelante. Así que lo siguiente también es completamente válido:

```
arn@localhost:~$ g++ libreria.cpp
libreria.h -c -o libreria.o
```

Lo que debemos hacer a continuación es enlazarlo al archivo fuente de inicio que creará el ejecutable.

```
arn@localhost:~$ g++ main.cpp libreria.o
-o programa
```

La ventana de ésta última opción es que podemos utilizar el archivo binario **librería.o** para enlazar con cualquier otro programa, por ejemplo, para utilizarlo en:

program2.cpp:

```
#include <iostream>
#include "libreria.h"
int main(int argc, char ** argv){
     std::cout << "El resultado de "
          << "82-23 es: " <<
(resta(82,23))
          << std::endl;
     return 0;
}
```

Se debe compilar de la siguiente forma:

```
arn@localhost:~$ g++ programa2.cpp
libreria.o -o programa2
```

Nótese que no fue necesario compilar la librería por segunda vez. De ésta forma, se crea el ejecutable programa2

```
arn@localhost:~$ ./programa2
El resultado de 82-23 es: 59
```

Creando una librería unificada

Hasta el momento, cada archivo, crea un archivo binario .o (object). Es posible unificar un conjunto de este tipo de archivos en una sola librería utilizando el comando ar.

```
arn@localhost:~$ ar cru libmatematicas.a
libreria.o libreria2.o
```

En la anterior orden, se creó un nuevo archivo llamado **`libmatematicas.a`**, cuyo objetivo es el de contener todos los archivos objeto necesarios. Vamos a crear un tercer programa ejecutable para probar nuestro nueva librería:

`programa3.cpp`

```
#include <iostream>
#include "libreria.h"
#include "libreria2.h"

int main(int argc, char **argv){
     int a = 67, b = 23;
     int s = suma(a,b);
     int r = resta(a,b);
     int m = multiplicacion(a,b);
     int d = division(a,b);
    std::cout <<" La suma de " << a << "
y "<< b
          << "=" << s << std::endl;
    std::cout <<" La resta de " << a <<"
y "<< b
          << "=" << r << std::endl;
    std::cout <<" La multiplicacion de "
<< a <<" y "<< b
          << "=" << m << std::endl;
    std::cout <<" La division de " << a
<<" y "<< b
          << "=" << d << std::endl;
 }
```

La forma de compilar éste nuevo programa es:

```
arn@localhost:~$ g++ programa3.cpp
libmatematicas.a -o programa3
```

Al ejecutar el programa debe observarse la siguiente salida:

```
arn@localhost:~$ ./programa3
La suma de 67 y 23=90
La resta de 67 y 23=44
La multiplicacion de 67 y 23=1541
La division de 67 y 23=2
```

Hasta aquí la primera parte del tutorial, en la segunda entrega tendremos algunas herramientas útiles como gdb, que es un compilador GNU, uso de make para compilación automatizada, librerías dinámicas, la gama de herramientas denominadas autotools, que son de utilidad cuando los proyectos tienen mucha más estructura en directorios y requerimientos de portabilidad.

Referencias

[1] http://sources.redhat.com/autobook

[2] http://www.gnu.org/prep/standards/

Autor

Arnold Marcelo Guzmán

Desarrollador
spacerockganimedes@gmail.com

Introducción a Django (3ra parte)

Django es un framework para el desarrollo de aplicaciones Web desarrollado en Python originalmente por Adrian Holovaty, Simon Wilson, Jacob Kaplan-Moss y Wilson Miner para World-Online el 2003 . Desde 2005 es software de código abierto (con una licencia BSD) y en septiembre de 2008 alcanzará la tan ansiada versión 1.0.

Introducción

En esta tercera entrega de la serie vamos a crear una primera aplicación que nos permitirá publicar artículos. Vamos a definir nuestros modelos, utilizando algunos de los campos que nos ofrece **Django** y algunas de las relaciones básicas y vamos a activar la interfaz de administración para nuestra nueva aplicación.

Creando una aplicación.

Django distingue entre un proyecto y aplicaciones. Un proyecto es la base de todo el desarrollo que albergará a las aplicaciones que pueden formar parte de uno o más proyectos.

Es más, aplicaciones pueden ser creadas independientemente de un proyecto y ser añadidas luego a otro proyecto, como el caso de la aplicación **`contact_form`** utilizada para el formulario de contacto en la segunda entrega de esta serie.

Django viene además con una serie de aplicaciones incluidas que nos permiten crear rápidamente un sitio y/o partes de un sitio, como flatpages, que introducimos en la primera entrega de esta serie.

Para crear nuestra primera aplicación utilizamos entonces **`manage.py`**

```
$ cd ~/Projects/atix
$ ./manage.py startapp articles
```

Este paso creará la estructura básica de nuestra aplicación: un subdirectorio con el nombre del proyecto (articles) dentro de nuestro proyecto y con tres archivos **`__init__.py`**, **`models.py`** y **`views.py`**.

- ✔ **`__init__.py`** es simplemente un archivo vacío que identifica a articles, nuestra aplicación, como un módulo en Python.
- ✔ **`models.py`** albergará los modelos de nuestra aplicación, es decir la descripción de nuestro datos, que será usada para crear las tablas necesarias en nuestra base de datos. Nos permitirá más adelante a través de una interfaz de programación manipular los objetos en la base de datos.
- ✔ **`views.py`** albergará las vistas necesarias para extraer los artículos de nuestra base de datos. Como veremos más adelante, es posible utilizar "vistas genéricas" para simplificar y acelerar el desarrollo de tareas repetitivas.

Modelos en Django.

Un modelo en **Django** es simplemente una clase derivada de la clase Model de **`django.db.models`**. Este modelo se traducirá luego en una tabla en la base de datos. Sus atributos corresponden o equivalen a los campos que tendrá nuestra tabla.

Además podemos definir métodos que tendrán nuestros objetos y formarán parte de la interfaz de programación (API) de nuestros objetos en el mapeador.

Nuestra aplicación tendrá dos modelos, categorías y artículos. Estableceremos una relación "muchos a muchos" entre los artículos y las categorías, es decir un artículo puede pertenecer a más de una categoría, y una relación de "uno a muchos", una llave foránea, para el autor de cada uno de los artículos, haciendo referencia al modelo de usuario existente en **Django**.

El modelo de categoría.

atix/articles/models.py

```
# encoding: utf-8
from django.contrib.auth.models import
User
from django.db import models
from datetime import datetime

class Category(models.Model):
    name = models.CharField(u'nombre',
max_length=32)
    slug = models.SlugField(u'slug',
max_length=32, unique=True)
    class Meta:
        ordering = ('name',)
        verbose_name = u'categoría'
        verbose_name_plural =
u'categorías'
    def __unicode__(self):
        return self.slug
    def get_absolute_url(self):
        return '/articles/category/%s/' %
self.slug
```

name es el nombre de nuestra categoría y es un campo de caracteres (una cadena) con un máximo de 32 caracteres. **slug** es un fragmento de la ruta que utilizaremos para identificar a nuestra categoría, por ejemplo "empezando-con-django".

El primer argumento de cada uno de los campos es el nombre que se utilizará en la interfaz de administración que veremos más adelante.

Django creará automáticamente un campo id en la tabla que identificará al objeto de manera inequívoca.

La clase interior Meta nos permite definir alguna propiedades adicionales de nuestro modelo, como ser el campo respecto al cual se ordenarán los objetos (ordering) y los nombres que aparecerán en la interfaz de administración (**verbose_name** y **verbose_name_plural**).

El modelo incluye además dos métodos muy simples, el primero, **__unicode__**, retorna simplemente una representación del objeto, que se utiliza en la interfaz de programación y en partes de la interfaz de administración.

El segundo método, **get_absolute_url**, retorna la ruta completa del objeto, algo que nos será útil cuando desarrollemos las plantillas de nuestra aplicación.

El modelo de artículo.

atix/articles/models.py

```
class Article(models.Model):
    author = models.ForeignKey(User,
related_name='articles',
        verbose_name=u'autor')
    title = models.CharField(u'título',
max_length=64, unique=True)
    slug = models.SlugField(u'slug',
max_length=64,
        unique_for_month='published')
    intro =
models.TextField(u'introducción')
    body = models.TextField(u'cuerpo')
    categories =
models.ManyToManyField(Category,
related_name='articles',
        verbose_name=u'categorías')
    published =
models.DateTimeField(editable=False,
default=datetime.now)
    last_updated =
models.DateTimeField(editable=False,
blank=True,
        null=True)
    class Meta:
        ordering = ('-published',)
        get_latest_by = 'published'
        verbose_name = u'artículo'
        verbose_name_plural =
u'artículos'
    def __unicode__(self):
        return self.title
    def get_absolute_url(self):
        return '/articles/%i/%i/%s/' % \
            (self.published.year,
self.published.month, self.slug)
    def save(self):
        if self.pk:
            self.last_updated =
datetime.now()
```

```
        super(Article, self).save()
```

author es una llave foránea, que hace referencia al modelo de usuario de **Django** (User), **related_name=articles** creará dentro de cada objeto de la clase User una lista con todos los artículos en los cuales aparece el usuario como autor. (Por defecto esta referencia tendría el nombre **article_list**.) La opción **verbose_name** define el nombre del campo que aparecerá en la interfaz de administración.

title, **slug** son campos similares a **name** y **slug** usados en el modelo de la categoría.

intro, **body** son ambos campos de texto, ésta vez sin límite de tamaño.

categories creará una referencia (a través de una tabla extra) "mucho a muchos". Cada objeto de la clase **Category** tendrá un una lista **articles** con todos los artículos publicados en esa categoría.

published, **last_updated** son campos de fecha y hora con un par de particularidades. Ambos no se mostrarán en la interfaz de administración (**editable=False**), el primero tiene por defecto el valor **datetime.now**, es decir el momento en que sea grabado (publicado) el artículo y quedará vacío hasta que se actualice (edite) el artículo.

La clase interior Meta y los métodos **__unicode__** y **get_absolute_url** son muy similares a los del modelo de categorías.

El método save extiende la funcionalidad del método save que por defecto tiene la clase Model, actualizando el valor del campo **last_updated** antes de grabar el objeto en la base de datos, si el objeto ya existe (**if self.pk**).

Activando la aplicación y creando las tablas.

Para activar nuestra nueva aplicación, es necesario añadir el nombre de nuestra aplicación a la lista de aplicaciones instaladas:

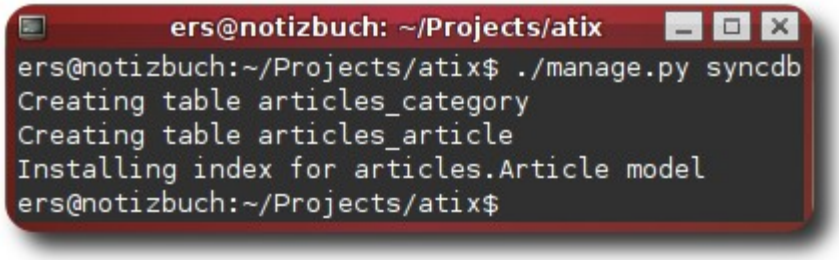
atix/settings.py

```
INSTALLED_APPS = (
    ...
    'articles',
)
```

Luego podemos crear las tablas en la base de datos usando una vez más manage.py, pero antes podemos revisar cómo se crearán las tablas:

```
ers@notizbuch: ~/Projects/atix
ers@notizbuch:~/Projects/atix$ ./manage.py sqlall articles
BEGIN;
CREATE TABLE "articles_category" (
    "id" integer NOT NULL PRIMARY KEY,
    "name" varchar(32) NOT NULL,
    "slug" varchar(50) NOT NULL UNIQUE
)
;
CREATE TABLE "articles_article" (
    "id" integer NOT NULL PRIMARY KEY,
    "author_id" integer NOT NULL REFERENCES "auth_user" ("id"),
    "title" varchar(256) NOT NULL UNIQUE,
    "slug" varchar(50) NOT NULL,
    "intro" text NOT NULL,
    "body" text NOT NULL,
    "published" datetime NOT NULL,
    "last_updated" datetime NULL
)
;
CREATE TABLE "articles_article_categories" (
    "id" integer NOT NULL PRIMARY KEY,
    "article_id" integer NOT NULL REFERENCES "articles_article" ("id"),
    "category_id" integer NOT NULL REFERENCES "articles_category" ("id"),
    UNIQUE ("article_id", "category_id")
)
;
CREATE INDEX "articles_article_author_id" ON "articles_article" ("author_id");
CREATE INDEX "articles_article_slug" ON "articles_article" ("slug");
COMMIT;
ers@notizbuch:~/Projects/atix$
```

Luego, podemos proceder a crear las tablas necesarias sincronizando la base de datos:



Interfaz de administración.

En la primera entrega utilizamos la interfaz de administración para crear nuestra primera página estática, ahora podemos utilizarla para nuestra aplicación. Para eso sólo es necesario crear un pequeño archivo con las opciones que tendrá nuestra interfaz. Inicialmente será simple.

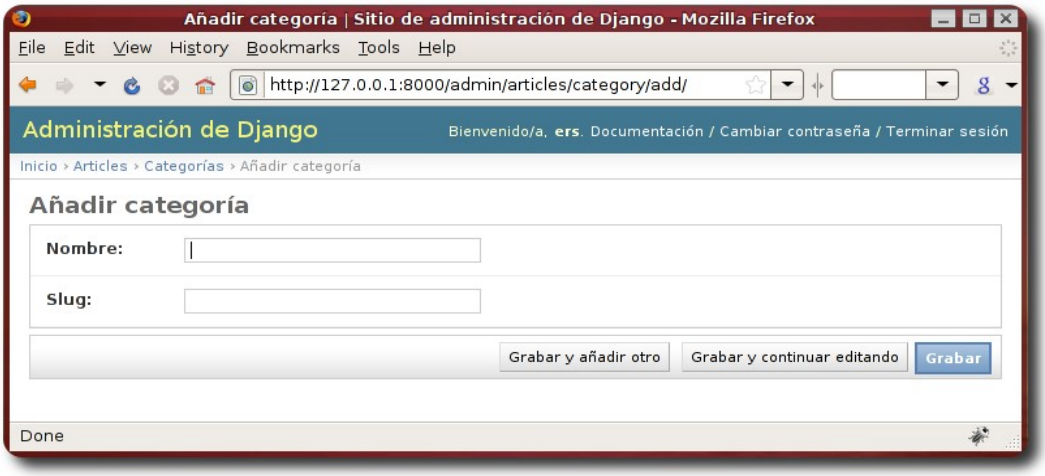
atix/articles/admin.py

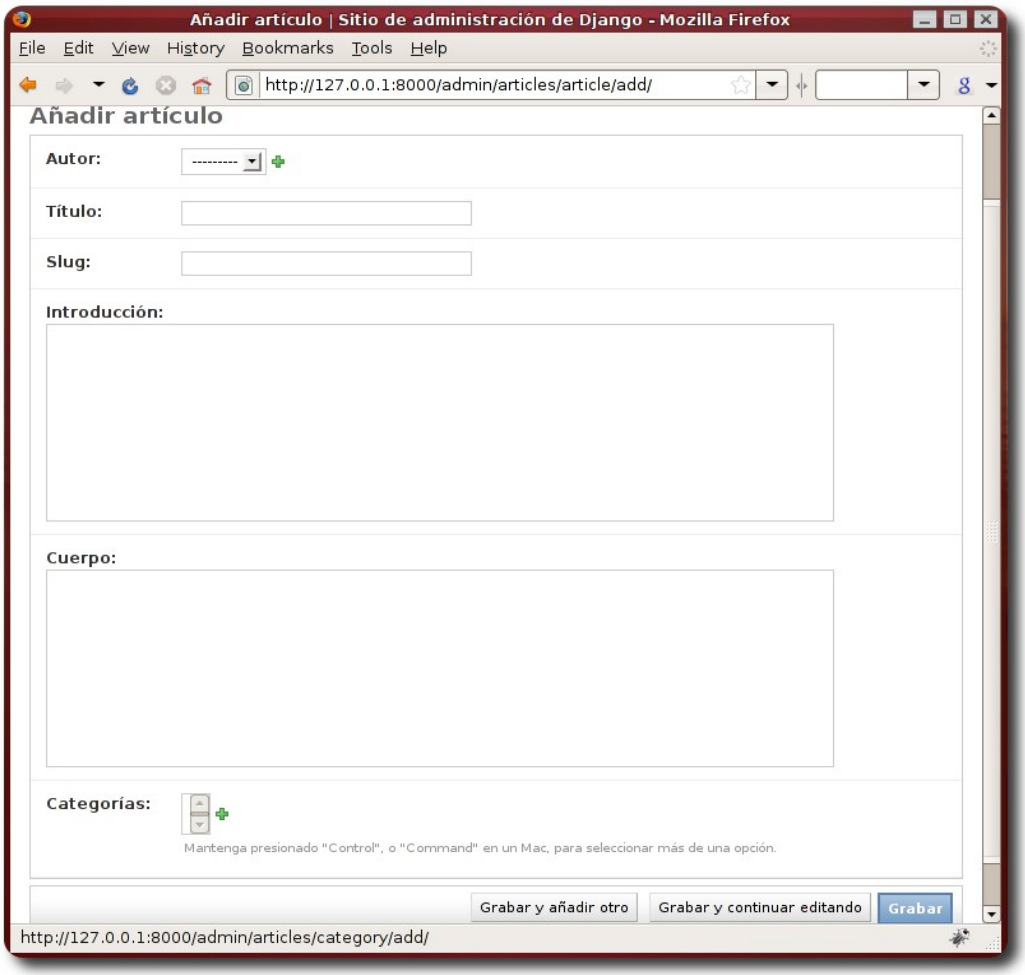
```
from django.contrib import admin

from models import Article, Category

admin.site.register(Article)
admin.site.register(Category)
```

Luego podemos iniciar nuestro servidor de desarrollo y dirigirnos a la interfaz de administración: **http://127.0.0.1:8000/admin/**, ahora podemos crear nuevas categorías, nuevos artículos, etc.





En la próxima entrega estaremos configurando más adecuadamente y personalizando la interfaz de administración, utilizaremos las vistas genéricas, crearemos una vista para extraer los artículos de la base de datos y desarrollaremos las plantillas.

Referencias

[1] http://www.djangoproject.com/

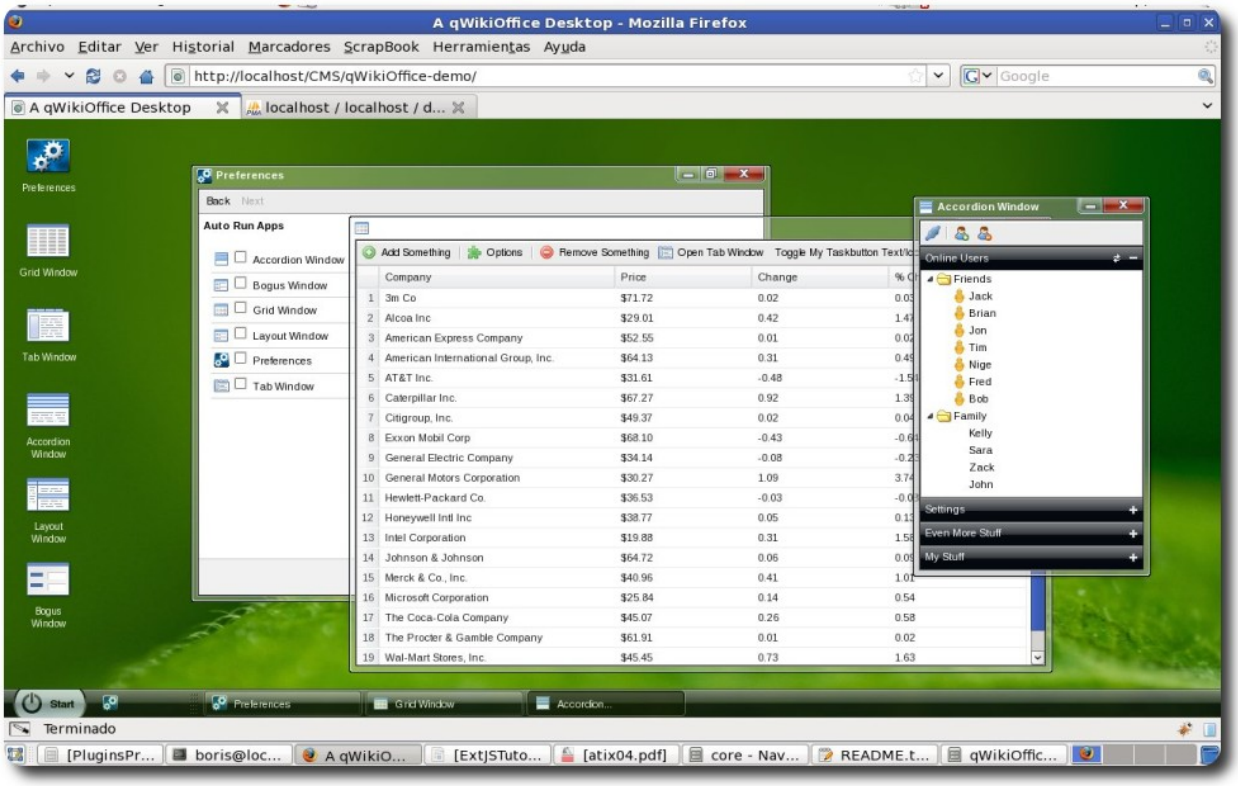
Autor

Ernesto Rico Schmidt
Usuario de Linux y Software Libre desde 1994
e.rico.schmidt@gmail.com

Introducción a Ext JS (1ra parte)

Ext JS es un Framework que permite crear aplicaciones Web 2.0 con interfaces muy similares a la de una aplicación de escritorio.

Introducción



Ext JS fue creado por Jack Sloqum a principios del 2006 conocido en esa época como YUI Ext debido a que utilizaba el Framework de Yahoo para funcionar; en Febrero de 2007 se lanza la primera versión de Ext 1.0 que necesitaba la librería YUI, meses después en Julio del 2007 se lanza la versión 1.1 la cual ya no necesitaba de YUI y permite que Ext JS se desarrolle como un proyecto independiente con adaptadores para poder funcionar con jQuery, YUI y Prototype, en Diciembre de 2007 se lanza la versión 2.0 con licencia LGPL3 y esto permite que la librería sea más conocida por empresas más grandes; en el lanzamiento de la versión 2.1 la licencia cambia a GPL3 lo cual provoca un revuelo en la comunidad, Ext JS ya que puede ser usado de forma libre y también se puede adquirir licencias comerciales del mismo. No es posible comparar a Ext JS con otras librerías de javascript ya que Ext está más orientado a AJAX que a Web, y me animo a decir de que no existe nada que se le compare porque no se puede encontrar algo tan completo y con la calidad de características que nos brinda este framework, actualmente. Ext JS se encuentra en la versión 2.2 distribuido con la licencia GPL3, muchas empresas como Abobe, SAP, Marketo y otras han desarrollado sus aplicaciones basadas en este Framework, es posible ver vídeos de

algunas aplicaciones en **http://extjs.com/blog**, y si alguien está por tomar la decisión de realizar una aplicación le recomiendo realizar la comunicación con el servidor usando JSON o XML olvidando el típico comportamiento Web inclusive AJAX basado en actualizaciones completas o parciales de HTML.

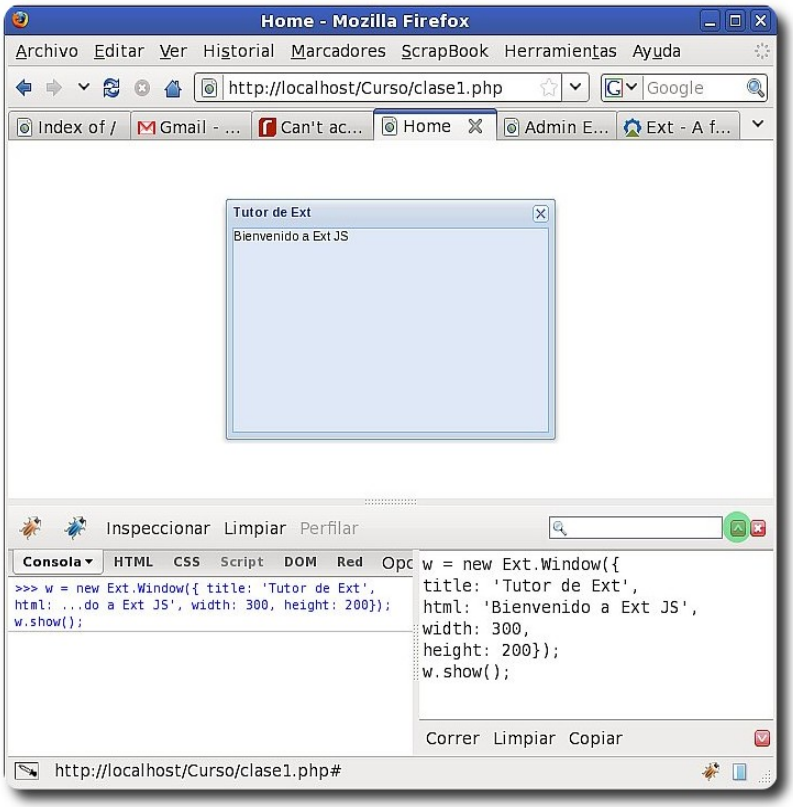
Ahora comencemos a programar, empezaré mostrando algunos elementos y como se utilizan los mismos, adicionalmente también es posible descargar un proyecto que realicé llamado archivaldo, realizado con CakePHP y Ext JS de la siguiente dirección **http://redecuanime.com/producto/archivaldo**

Descargando e instalando Ext JS

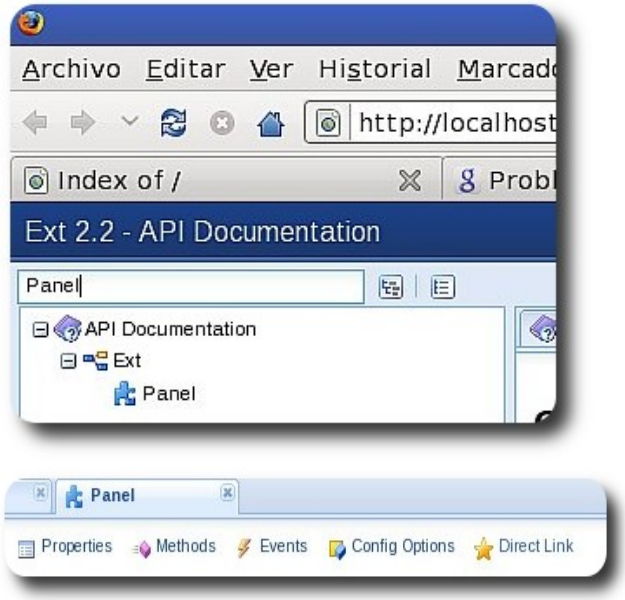
Para iniciar hay que descargar Ext JS de **http://extjs.com**, (al momento de éste tutorial, la última versión de esta librería es la versión 2.2) otras herramientas indispensables para poder desarrollar con Ext son el navegador Firefox que lo puedes descargar de **http://mozilla.org** y el addon Firebug para Firefox que lo puedes descargar de **http://getfirebug.com**, una ves descargado Ext descomprime en algún lugar que tenga acceso tu servidor Web, al descomprimir se crea la carpeta ext-2.2 la cual contiene el código JavaScript, imágenes y estilos CSS para que pueda funcionar, debes considerar que Ext JS no solo es una librería de JavaScript sino de CSS del cual depende para poder crearse los distintos componentes y el cual puede ser modificado para poder darle otra apariencia a los componentes. Ahora debes crear un archivo html en algún lugar que pueda acceder tu servidor, con la siguiente cabecera:

```
<head>
<link rel="stylesheet" type="text/css" href="ext-2.2/resources/css/ext-all.css" />
<script src="ext-2.2/adapter/ext/ext-base.js"></script>
<script src="ext-2.2/ext-all-debug.js"></script>
</head>
```

Con la ayuda de firebug puedes correr el código que aparece en la imagen.



Podemos ver que el código nos muestra una ventana la cual la puedes arrastrar o redimensionar, en la parte inferior izquierda firebug te muestra lo que se ha ejecutado y en la parte inferior derecha es donde tu puedes escribir el código que después puedes ejecutar haciendo click en "*Correr*" o presionando las teclas "*Ctrl+Enter*". Básicamente una ventana es un Panel, si ustedes verifican la documentación de Ext que se encuentra en directorio docs ahí



podrán ver los distintos objetos que éste tiene, un componente Ext se compone de propiedades, métodos y eventos, en Ext es posible crear eventos, la documentación es amplia y de excelente calidad, si tienen consultas pueden acceder al foro en : "**`http://extjs.com/forum`**" o al canal de IRC en Freenode "extjs", en esos dos lugares encontrarán tanto información como respuestas.

Para que puedan seguir este tutor vean la excelente documentación que provee Ext JS, dentro de la carpeta ext-2.2, está la librería JavaScript, css, imágenes, código fuente y los documentos. En la carpeta docs, se encuentra la documentación y se tiene la posibilidad de realizar búsquedas, como se muestra en la imagen, donde también se aprecia:

- ✔ **Properties** que son las propiedades del Panel
- ✔ **Methods** que son las funciones del Panel
- ✔ **Events** son los eventos del Panel.

```
pan = new Ext.Panel({title:'Prueba', html:'Hola'});
pan.on('add', function(panel, com, i){
   alert('Componente nuevo adicionado en pos '+i);
}, panel);
```

El código anterior crea un Evento que observa si se adicionan nuevos componentes a un Panel y muestra una alerta en que posición ha sido adicionado éste nuevo componente. Ahora vayan a la documentación del Panel hagan click en Events y busquen el evento add, el cual nos muestra el siguiente texto

```
   add : ( Ext.Container this, Ext.Component component, Number index )
```

Lo que nos dice el mismo, es que cuando se produce el evento add se van a enviar estos argumentos, el primero es el contenedor *Ext.Container,* en éste caso el panel que hemos creado, luego el componente que se está adicionando al panel que podría ser otro panel, un formulario u otro componente, y para finalizar el índice (*Number*) en el cual se está adicionando el componente; luego tenemos los *Config Options* que son las opciones iniciales de configuración como son la dimensión (*width, height*), título (*title*) y otros dependiendo del componente y para finalizar tenemos un *Direct Link* que solo es un vínculo.

Pequeña muestra

El siguiente código crea una pequeña muestra (un layout con varios Paneles), que a continuación explicaremos con detalle.

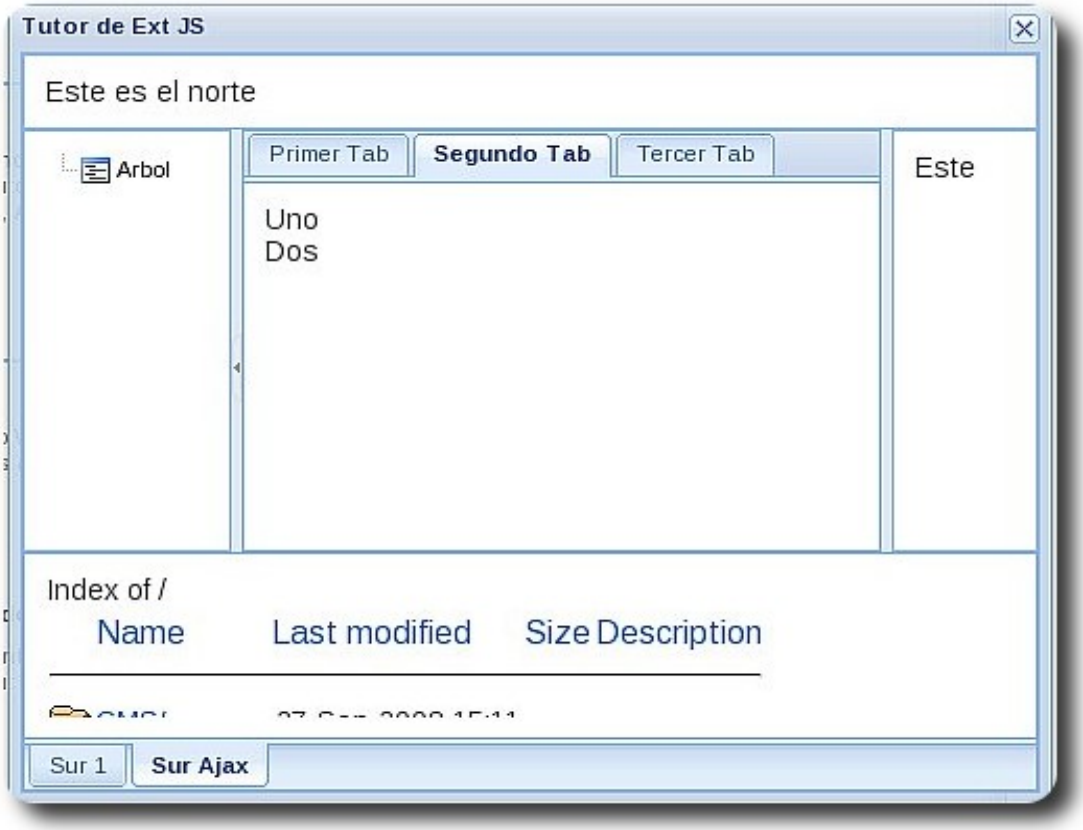
```
w = new Ext.Window({
    title: 'Tutor de Ext JS',
    width: 500,
    height: 400,
    expandible: true,
    layout: 'border',
    defaults: {
        bodyStyle: 'padding: 10px; font-size: 14px', split: true
    },
    items: [{
        xtype: 'panel', region: 'north', html: 'Este es el norte', split: false
    },{
        xtype: 'treepanel', region: 'west', width: 100, collapsible: true,
        collapseMode: 'mini', bodyStyle: 'padding: 0px',
        root: new Ext.tree.AsyncTreeNode({
            text: 'Arbol', children: [{text:' Nodo 1', singleClickExpand: true,
children: []},
                { text: 'nodo 2', singleClickExpand: true,  children: []}]
        })
    },{
        xtype: 'tabpanel', region: 'center', activeTab: 1, title: 'Centro',
        items: [{
      title: 'Primer Tab', html: 'Contenido del Primer Tab'
        },{
      title: 'Segundo Tab', html : '<ul><li>Uno</li><li>Dos</li></ul>'
        },{
      title: 'Tercer Tab', html: '<table border="1"><tr><th>Título</th></tr></table>'
    }]
    },{
        region: 'east', html: 'Este', width: 70
    },{
        xtype: 'tabpanel', region: 'south', collapsible: true,
        tabPosition: 'bottom', activeTab: 0, height: 120,
        collapseMode: 'mini', defaults: {autoScroll: true,},
        items: [{
      title: 'Sur 1', html: 'Contenido Tab Sur 1'
        },{
      title: 'Sur Ajax', html : 'Soy del sur',
      autoLoad: {url: 'http://localhost', method:'GET', params: {id: 1, tema:
'nuevo'}}
        }]
    }]
});
w.show();
```

Primero hemos creado una nueva Ventana usando **`new Ext.Window({`**, definimos sus propiedades, como **`width, height, title`**, etc., indicamos el layout (**`layout: 'border'`**), éste básicamente se compone de 5 regiones (**`north, west, center, east, south`**), lo siguiente que realizamos es adicionar items, que pueden ser formularios, paneles, tabpanels, grids, árboles u otros; posteriormente definimos algunas propiedades por defecto: **`{ bodyStyle: 'padding: 10px; font-size: 14px', split: true}`** que permiten definir propiedades de los nuevos componentes que vayamos adicionado, en nuestro caso son los siguientes:

- ✔ En la región del norte *north* hemos adicionado un Panel (es el componente por defecto que se adiciona), no tiene título y solo tiene un texto para el html.
- ✔ En la región oeste *west* hemos adicionado un árbol con un solo nodo, hemos definido en el *bodyStyle*: que el padding sea 0px, en Ext generalmente los paneles (GridPanel,

Panel, Tab Panel y otros) tienen la cabecera que es donde se encuentra el título y el cuerpo donde se encuentra el contenido. En el caso de este panel hemos adicionado 2 nodos children a la raíz root

- ✔ En el panel central tenemos un TabPanel, para el cual hemos definido que por defecto muestre el panel en la segunda posición (*activeTab: 1*). Básicamente cada tab es un Panel (*Ext.Panel*) y el *TabPanel* es un contenedor de éstos, también es posible adicionar otro tipo de paneles
- ✔ En el panel este *east* solo tenemos un panel que nos sirve para mostrar el layout
- ✔ Por último en el sur *south*, tenemos un tabPanel con dos elementos: uno de los tabs puede cargar mediante Ajax el contenido (`autoLoad: {url: 'http://localhost', method: 'GET', params: {id:1, tema: 'nuevo'}}`) al cual le indicamos el método de la consulta GET y pasamos 2 parámetros (id y tema), además que indicamos que los tabs se muestren en la parte inferior *tabPosition: 'bottom*'



Las ventanas creadas con el layout: '*border*', pueden funcionar solo con algunas regiones, pero es necesario que exista la región '*center*', les sugiero ir probando el manejo de regiones del panel, basándose en la documentación para tener mayor referencia

En un siguiente número les mostraré la manipulación de algunos componentes, adicionar eventos y la programación de algunas acciones.

En **`http://boliviaonrails.com`**, podrán encontrar un tutorial donde describo como manipular un árbol y como integrarlo con Ruby on Rails

Agradecimiento

Gracias por el espacio que me provee la revista ATIX para este artículo, en un próximo número espero poder presentarles más facilidades y particularidades de este extenso Framework.

Referencias

[1] http://extjs.com

[2] http://getfirebug.com

Autor

Boris Barroso
boriscyber@gmail.com

Desarrollo Ágil con Ruby on Rails (1ra Parte)

Ruby on Rails es un framework de desarrollo web ágil, elaborado por David Heinemeier Hansson, que el 2004 lanzó la versión pública a partir de desarrollar el proyecto Basecamp, Ruby on Rails (RoR) está desarrollado en el lenguaje de programación Ruby, RoR actualmente se encuentra en la versión 2.1.1 en la que contribuyeron más de 1400 desarrolladores del mundo con 1600 parches al framework, por éstos datos RoR es uno de los proyectos open source con más movimiento actual.

Introducción

Actualmente dentro de este mundo más competitivo se exigen proyectos informáticos a una velocidad impresionante, frases como "*Lo quiero para ayer*", "*El costo es excesivo*" son el día a día del desarrollo de software; entonces tenemos dos recursos que debemos explotar al máximo que son: el tiempo y costo. Una alternativa actual que se van imponiendo al problema son:

- ✔ Los lenguajes dinámicos.
- ✔ Los frameworks de desarrollo.

Los anteriores se están haciendo bastante comunes al desarrollar aplicaciones, dentro de los lenguajes dinámicos apareció Ruby, un lenguaje inspirado en SmallTalk creado por Matz y en Frameworks está Rails, ambos combinados crean a **Ruby on Rails**.

Ruby on Rails, es la plataforma de desarrollo ágil de proyectos enfocados en la web; por el 2005 se hizo popular asociar la web 2.0 con **Ruby on Rails**. RoR utiliza el Patrón MVC (Modelo Vista Controlador) Que ya lo debes conocer, y que permite ser más ordenado al momento de desarrollar una aplicación web, tiene un sistema de plugins que permite reutilizar código ya desarrollado, maneja a perfección en tema de REST lo cual hace que toda tu aplicación sea un repositorio de recursos (html, xml, pdf, etc), generación de código (scaffold), Ajax, etc, en fin es hoy por hoy el máximo representante dentro de frameworks de desarrollo; la lista top 100 de sitios hechos en rails la puedes ver en el sitio de internet: **`http://rails100.pbwiki.com/Compete+Rankings`**, su uso se está incrementando rápidamente por lo fácil y robusto desarrollo que ofrece.

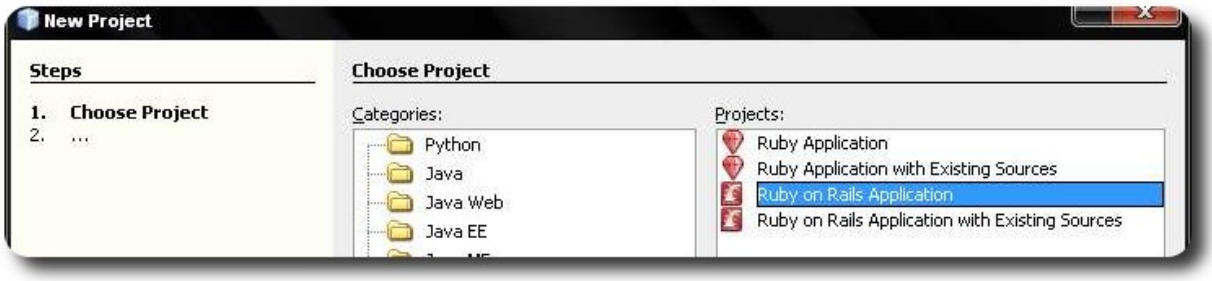
Que precisamos

Para desarrollar aplicaciones en rails precisamos tener instalado:

- ✔ Interprete ruby (1.8.7)
- ✔ Ruby-gems(1.3)
- ✔ Rake (0.8.3)
- ✔ Rails (2.1.1)
- ✔ Mongrel (opcional 1.1.5).

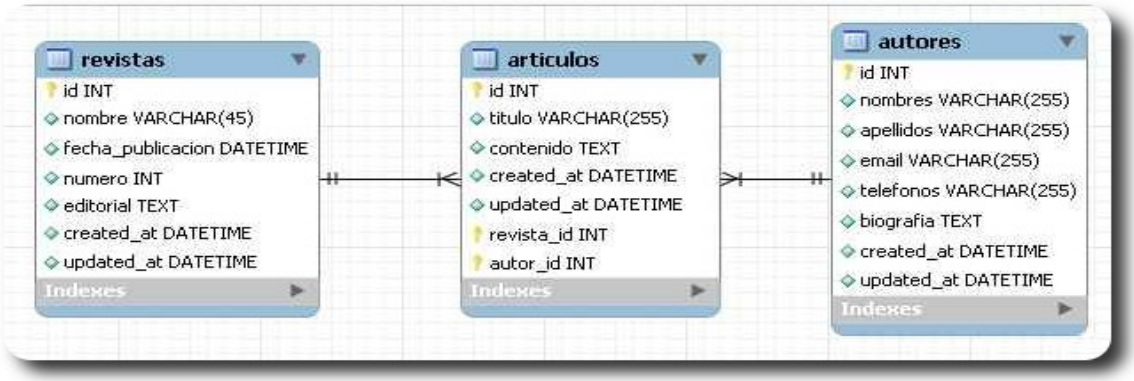
Entorno de desarrollo

Bien empecemos, para crear una aplicación Rails y para poder trabajar el día a día utilizaremos un IDE, en éste caso Netbeans 6.5 Beta, que es un excelente IDE de desarrollo, y si no sabes cómo instalar Rails ya viene todo pre instalado pero en la versión Java: Jruby, puedes asimismo manejar varios intérpretes de ruby dentro de Netbeans y sus versiones sin tener problemas. Ahora empecemos el tutorial, en el menú de Netbeans: **`File -> New Proyect`** , con eso se iniciará el asistente que nos guiará paso a paso en la creación de la aplicación, seleccionar el tipo de proyecto **Ruby on Rails**:



Nuestra aplicación

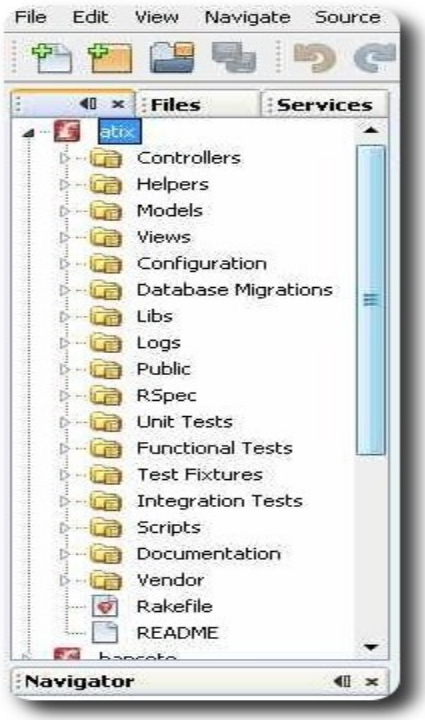
Antes de proseguir les mostraré el diagrama E-R para que tengan una idea de nuestra aplicación:



Como ven es una relación simple una revista tiene uno o más artículos y cada autor escribe uno o más artículos.

Estructura del proyecto

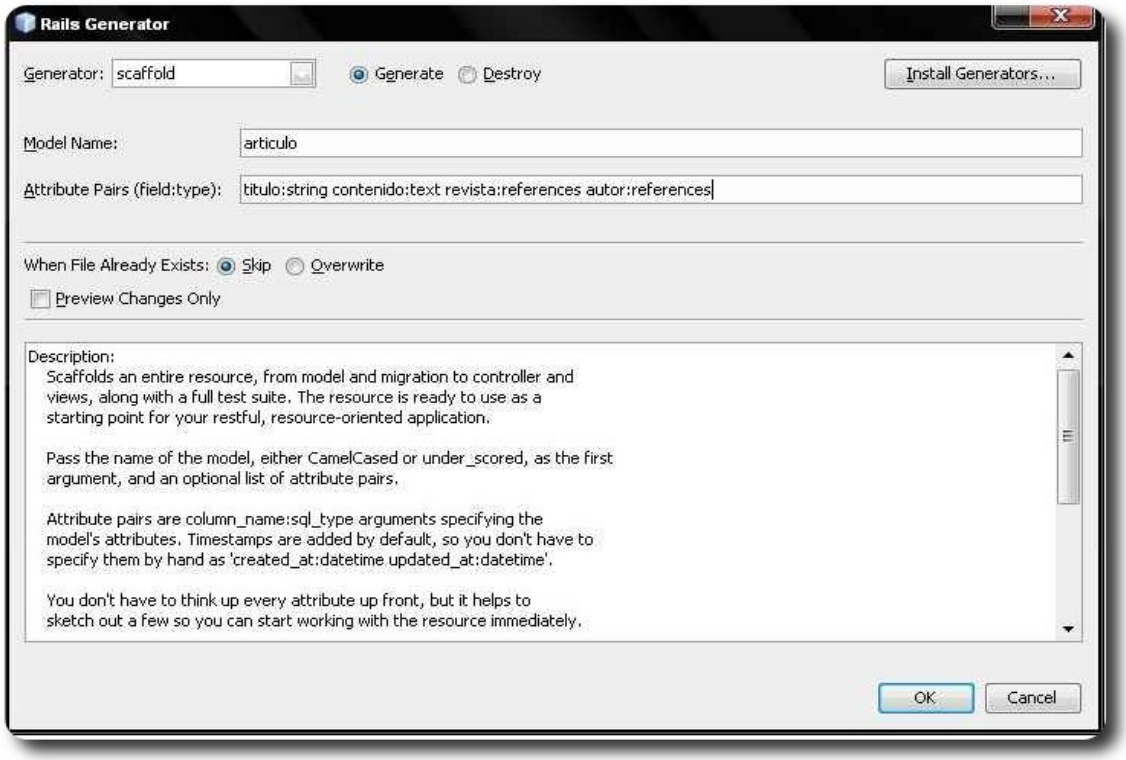
Seguimos los pasos del wizard, dando el nombre del proyecto (Atix) y algún otro dato, finalizando este proceso, se creará el esqueleto de la aplicación, tal como muestra la figura.



Dentro de la estructura de un proyecto, existen directorios y archivos que nos permiten adecuar aspectos como:

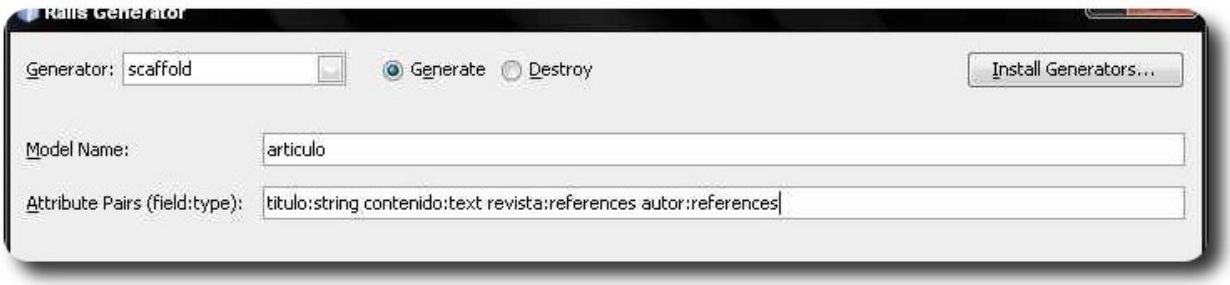
- ✔ **Modelos,** adecuar relaciones entre los modelos del proyecto
- ✔ **Controladores**, adecuar la lógica y flujo del proyecto
- ✔ **Rutas,** adecuar la presentación y acceso de la URL
- ✔ **Plantillas**, adecuar la presentación visual del proyecto o parte del mismo
- ✔ **Helpers**, segmentos de código con funcionalidades especificas.
- ✔

Haciendo click derecho en la raíz del proyecto, en éste caso atix con el logo de rails, mostrará un menú donde elegimos la primera opción que es Generate…, que permite hacer un scaffold (andamio) de nuestra aplicación, antes esta funcionalidad recogía de la misma tabla los campos, pero ahora es mucho más versátil, como se muestra en la figura.



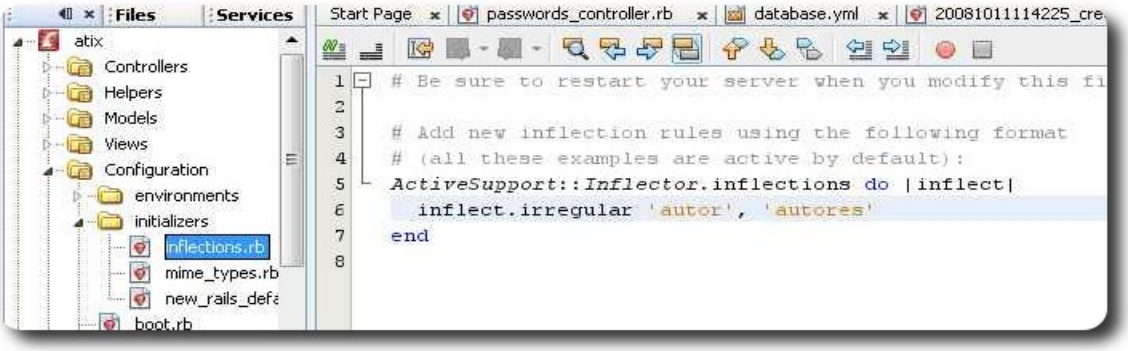
Aquí vemos que generamos el modelo revista, que será físicamente la tabla de base de datos: revistas, esto por convención el modelo en singular y la tabla en plural, luego indicamos los campos de éste modelo con el par nombre: tipo, en este caso será: un nombre como string, la fecha_publicación como una fecha, el número de la revista como un entero y un campo de texto editorial para la editorial.

Bien con esto ya hemos creado una serie de archivos que iremos detallando poco a poco, pero por el momento crearemos el modelo artículo de forma análoga, aunque con ciertas variantes como vemos en la figura.

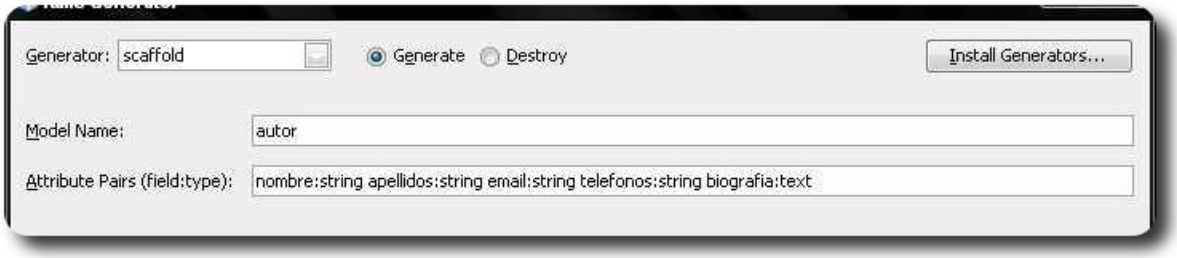


Dentro de este usual scaffold, vemos una nueva característica que es el atributo *references* que indica que ésta tabla tiene una relación con otra, definimos una relación del tipo: una revista tiene varios artículos y que un autor tiene varios artículos.

Ahora deberíamos generar el modelo de autores, pero aquí hay un pequeño problema: el plural de autor es autores, Rails detecta automáticamente el plural de los modelos en ingles, pero no en algunos en español, tal como autor-autores, para esto editaremos un archivo en **`config/initialicers/inflections.rb`** , como muestra la figura.



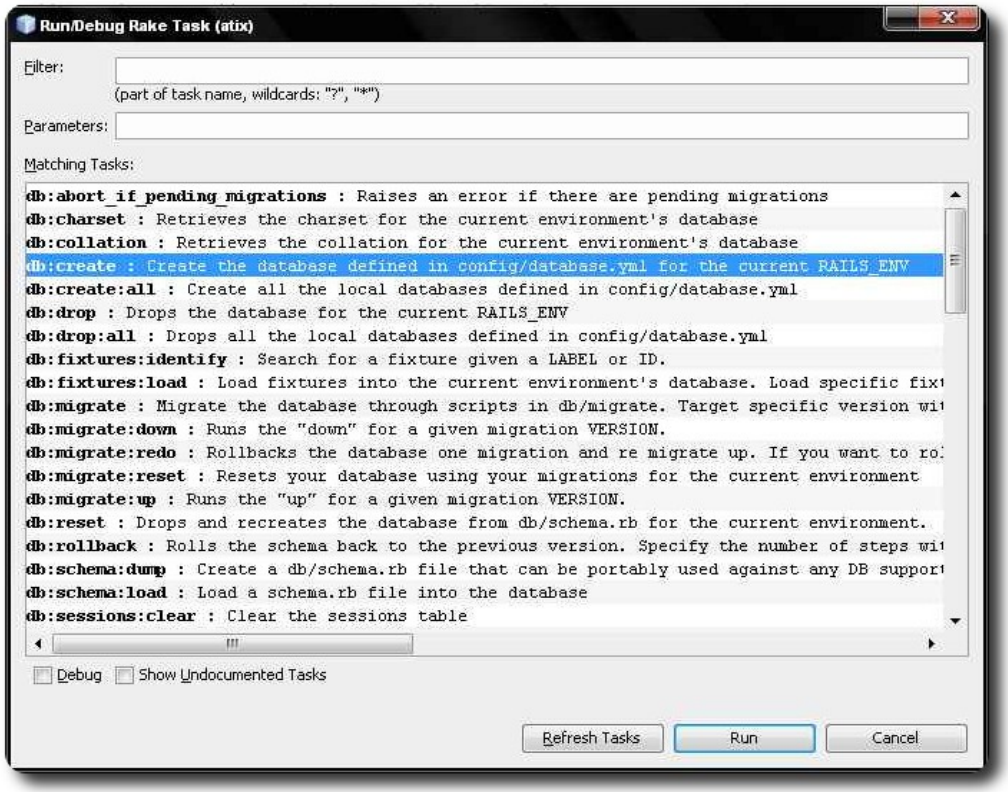
Basta con indicar a Rails nuestra nueva conjugación, y realizamos un nuevo Generate.



Ya tenemos nuestra base de la pequeña aplicación. ahora haremos que funcione, primero creamos y migramos la base de datos, para esto editamos los datos necesarios y correctos para la conexión a nuestra base de datos, en **`/config/database.yml`**.

Ahora realizamos la tarea rake para crear la base de datos:

`Rake db:create` o en la raíz click derecho y elegimos **`Run/Debug rake task`**… en el cual nos mostrará lo siguiente:



Seleccionando Run, creará nuestra base de datos, luego migramos el esquema de la base de datos: en raíz del **`proyecto -> Migrate database -> to Current version`**, con esto crearemos la base de datos y crearemos las tres tablas.

Modificando rutas

Ahora haremos unos pequeños cambios en las rutas, editamos el **`config/routes.rb`**

```
ActionController::Routing::Routes.draw do |map|
  map.resources :autores
  map.resources :articulos
  map.resources :revistas
  map.root :controller => "revistas"
  map.connect ':controller/:action/:id'
  map.connect ':controller/:action/:id.:format'
end
```

Nodificando plantillas

Eliminamos el **`/public/index.html`** y creamos el archivo que será la plantilla general de la aplicación, creamos **`/app/views/layouts/application.html.erb`** y escribimos lo siguiente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html;charset=UTF-8" />
    <title>Articulos: <%= controller.action_name %></title>
    <%= stylesheet_link_tag 'scaffold' %>
  </head>
  <body>
    <%= link_to("Revistas", revistas_path) %> | <%= link_to("Articulos",
articulos_path) %> | <%= link_to("Autores", autores_path) %>
    <p style="color: green"><%= flash[:notice] %></p>
    <%= yield  %>
  </body>
</html>
```

Y eliminamos los archivos restantes dentro de la carpeta layouts (**`artículos.html.erb`**, **`autores.html.erb`** y **`revistas.html.erb`**) e iniciamos la aplicación rails (click derecho al proyecto y Run).

Con esto tenemos la aplicación corriendo, podemos realizar varias tareas (CRUD o ABM) como ver, añadir, editar, eliminar los registros fácilmente. Antes de terminar tenemos que editar la vista de artículos para que muestre la relación con las otras tablas.

En **/app/views/artículos/new.html.erb**

```
<h1>New articulo</h1>
<% form_for(@articulo) do |f| %>
 <%= f.error_messages %>
 <p>
  <%= f.label :titulo %><br />
  <%= f.text_field :titulo %>
 </p>
 <p>
  <%= f.label :contenido %><br />
  <%= f.text_area :contenido %>
 </p>
 <p>
  <%= f.label :revista %><br />
  <%= f.select :revista_id, Revista.all.collect{|x| [x.nombre, x.id]} %>
 </p>
 <p>
 <%= f.label :autor %><br />
 <%= f.select :autor_id, Autor.all.collect{|x| [x.nombre + " " + x.apellidos, x.id]} %>
 </p>
 <p>
   <%= f.submit "Create" %>
 </p>
<% end %>
<%= link_to 'Back', articulos_path %>
```

Las líneas modificadas nos permiten seleccionar las revistas disponibles y los autores registrados, lo que permitirá mostrar un selector de opciones, primero para elegir la revista y otro para elegir el autor de éste artículo, en ésto utilizamos una expresión lambda, que crea un array de opciones a partir de los registros de la tabla.

Ahora dejemos para una segunda parte más funcionalidad, validaciones de campos, trabajo con las relaciones de las tablas desde código, Ajax, etc.

Referencias

[1] http://www.rubyonrails.org/

[2] http://www.rubyonrails.org.es/

Autor

Carlos Ramos
Lic. Informática UMSA
Lider de Wiebia, soluciones web 2.0
carakan@gmail.com

Trac: Gestión de proyectos de desarrollo de Software

La gestión de proyectos de desarrollo de software, es un elemento imprescindible al momento de encarar proyectos de desarrollo, porque ésto implica considerar tópicos como: control de versiones, wikis, manejo de bugs, etc. En la actualidad existen varias opciones para éste fin, pero una de las que destaca en el mundo del software Libre es **Trac,** por su sencillez, facilidad y por su calidad.

Introducción

La gestión de proyectos de desarrollo de Software, hoy por hoy se ha convertido en un elemento indispensable en el proceso de desarrollo de un producto software.

La gestión de proyectos es un aliado importante de la ingeniería de software en la tarea de conseguir como resultado un producto de software de calidad.

Gestión de Proyectos

La gestión de proyectos precisa contemplar con algunas herramientas como:

- ✔ Sistema de planificación
- ✔ Sistema de gestión documental
- ✔ Sistema de control de versiones
- ✔ Sistema de gestión de incidencias

Sistema de planificación

Objetivo

- ✔ Permitir organizar el proyecto en función de hitos, tareas, subtareas, asignación y control de tiempos, recursos materiales y humanos.
- ✔

Idealmente

- ✔ Permitir hacer el seguimiento y reajustar la planificación en función de la evolución del proyecto.

Recomendación

- ✔ Disponer de herramientas para llevar el control de tiempos estimados y empleados para cada tarea; para poder controlar la evolución del proyecto.
- ✔ Es importante que las personas inmersas en el proyecto deban reportar el tiempo que dedican a cada tarea y actualicen el estado de las mismas con relativa frecuencia (recomendable diariamente)

Sistema de gestión documental

Objetivo

- ✔ Servirá para almacenar y mantener los documentos obtenidos o generados durante el desarrollo del proyecto y acceder a ellos cómodamente.
- ✔ Cada hito, tarea o subtarea implica la obtención o generación de documentación (actas de reuniones, documentos de diseño, etc.).

Idealmente

- ✔ Debe permitir que almacenemos esa documentación en el propio sistema.

Sistema de control de versiones

Objetivo

- ✔ Permitir el desarrollo concurrente para mantener la historia del código fuente y parte de la documentación producida en el proyecto.
- ✔ Al tratarse de proyectos informáticos, lo normal es que se trabaje con código fuente y con documentos que van evolucionando a lo largo del desarrollo y que deben ser modificados por múltiples personas.

Idealmente

- ✔ Disponer de un sistema de control de versiones que permita mantener la historia de los ficheros generados y que más de una persona trabaje concurrentemente sobre el mismo código.

Sistema de gestión de incidencias

Objetivo

- ✔ Permitir hacer el seguimiento de los errores detectados y sus correcciones, tanto aquellos reportados por los responsables de la prueba del software como por los desarrolladores o los usuarios normales.
- ✔ También se puede utilizar como sistema de seguimiento de tareas de corta duración asociadas a fases del proyecto, a errores detectados o a cambios relacionados con solicitudes de mejora solicitadas por el cliente.
- ✔

Idealmente

- ✔ Deberá controlar todas y cada una de las posibles incidencias que puedan ocurrir en el desarrollo de un producto software.

Qué es Trac?

- ✔ **Trac** es un sistema que integra varios componentes con capacidades suficientes para la gestión de proyectos de desarrollo de software.
- ✔ **Trac** es un sistema web multiplataforma ligero y extensible.
- ✔ **Trac** es un programa pensado para desarrolladores que necesitan mantener un proyecto. Programado en python y ejecutado a través de mod_python o como cgi o fastcgi usando un servidor web, permite llevar una serie de utilidades propias para un proyecto.

Funcionalidades

- ✔ **Wiki:** Empleado para documentar cualquier aspecto del proyecto de modo colaborativo y sin necesidad de herramientas especiales.
- ✔ **Planificación (Roadmap):** Sistema para definir y visualizar el estado de los hitos de un proyecto (un hito incluye una descripción y una fecha y se usa como atributo de los tickets, que se asocian a hitos concretos).
- ✔ **Manejo de eventos (Timeline):** Sistema de seguimiento de eventos en el sistema:
  - ✔ Histórico de cambios en el wiki
  - ✔ En el sistema de control de versiones,
  - ✔ En el sistema de gestión de incidencias o vencimiento de un hito
- ✔ **Búsquedas:** Permite localizar páginas del wiki, comentarios dentro de los conjuntos de cambios o tickets en los que aparece una palabra.

- ✔ **Visor de Código:** Integrado con algún sistema de control de versiones (asociado al proyecto), nos permite ver los cambios que se han producido en el programa de una forma visual (estado actual del repositorio, los cambios que se han ido produciendo, comparar distintas versiones de ficheros en línea, etc). **Trac** únicamente es un interfaz del repositorio.

Componentes y características adicionales

Trac ha sido concebido de forma modular donde se pueden añadir plugins que proporcionan distintas funcionalidades. (casi todos los componentes estándar son módulos que pueden ser activados, desactivados o reemplazados o modificados por otros). Entre sus características adicionales se encuentran:

- ✔ **Administración:** Personalización de entorno, manejo de usuarios, permisos, plugins, etc.
- ✔ **Autenticación:** LDAP, BBDD o fichero.
- ✔ **Uso de VCS:** Subversión, Bazaar, GIT, Mercurial o Monotone.
- ✔ **Servicios adicionales:** blogs, foros, etc.

Requisitos de instalación

Los requisitos de las versiones actuales (0.11.x) son:

- ✔ Python, por estar desarrollado en python, además precisa los enlaces (bindings) con algunos de los subsistemas que emplea, como por ejemplo: SQLite o Subversión.
- ✔ Sistema de proceso de plantillas ClearSilver, actualmente reemplazado por Genshi.
- ✔ Soporte de la BBDD que vayamos a utilizar (SQLite, PostgreSQL o MySQL).

Instalación de Trac

La instalación depende de la distribución que se utilice:

- ✔ En el caso de CentOS o algún otro clon de RHEL: `yum install trac`
- ✔ En el caso de Ubuntu: `apt-get install trac`

Estructura de un proyecto de Trac

Cuando procedemos a la creación de un Proyecto en **Trac**, éste crea una estructura como se muestra en la figura.

Gráfico 1: Estructura de un proyecto de **Trac**

Creando un proyecto

Se emplea un programa que se invoca desde la línea de comandos y que genera una estructura de directorios con la configuración del proyecto y los ficheros relacionados, así como muestra la figura:

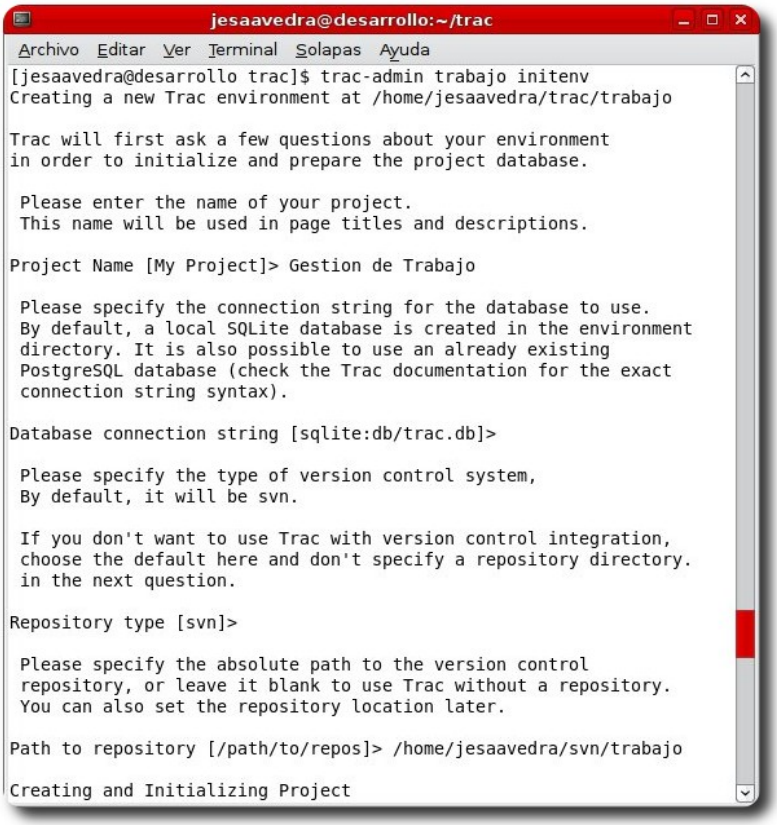


Gráfico 2: Creación del proyecto

Nota: Si deseamos que **Trac** interactúe con Subversión, o una BBDD distinta de SQLite será necesario crear el repositorio y/o la BBDD según corresponda antes de crear el proyecto).

Ejecución y funcionamiento de Trac

Trac puede funcionar de dos formas:

- ✔ Mediante su propio servidor (tracd)
- ✔ Mediante un servidor estándar (lighttpd, apache2) que tenga soporte para ejecutar código python usando scripts de CGI, FastCGI o mod_python.

En esta primera parte del tutorial ejecutaremos **Trac** mediante su propio servidor, las instrucciones de ejecución y acceso a **Trac** se muestran al finalizar la creación del proyecto.

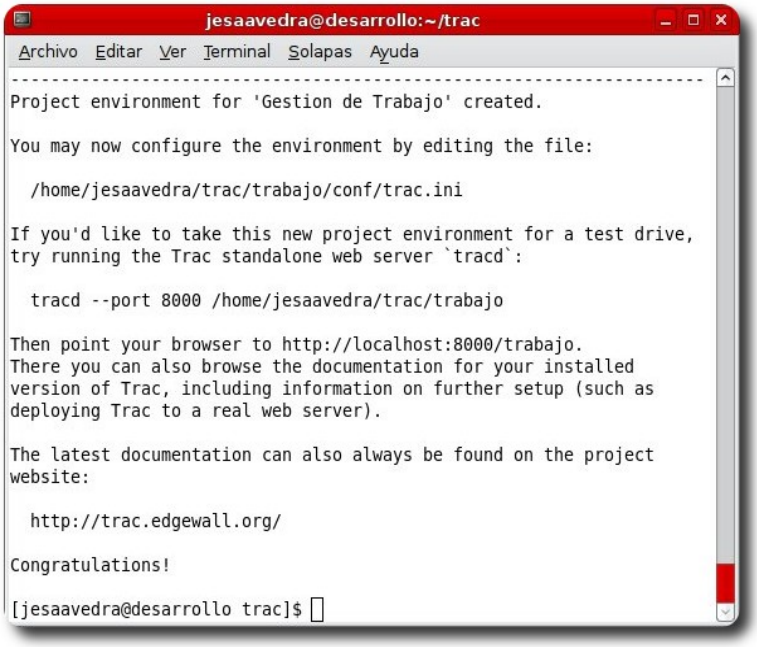


Gráfico 3: Detalles de ejecución y acceso

Ejecución y pruebas

Una vez que **Trac** se ejecute podemos empezar a probar todas y cada una de las características provistas: wiki, timeline, roadmap, visor de código fuente, visualización de tickets, búsquedas.

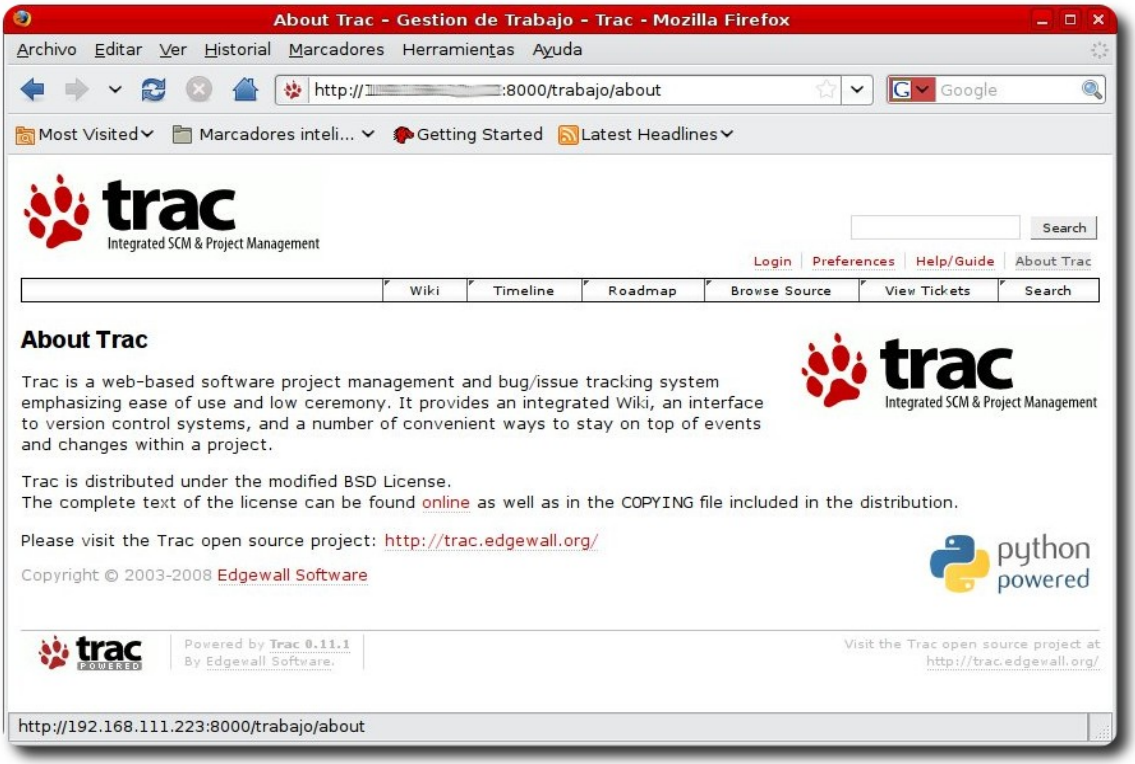


Gráfico 4: Acerca de **Trac**

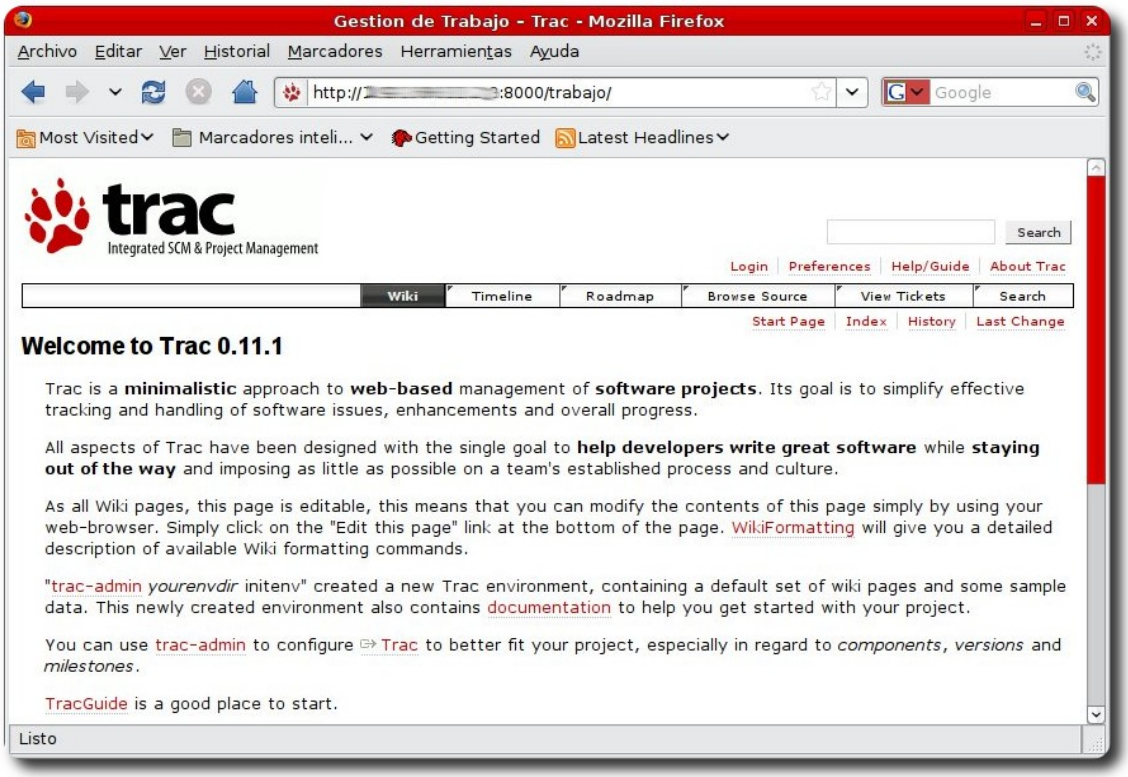


Gráfico 5: Wiki

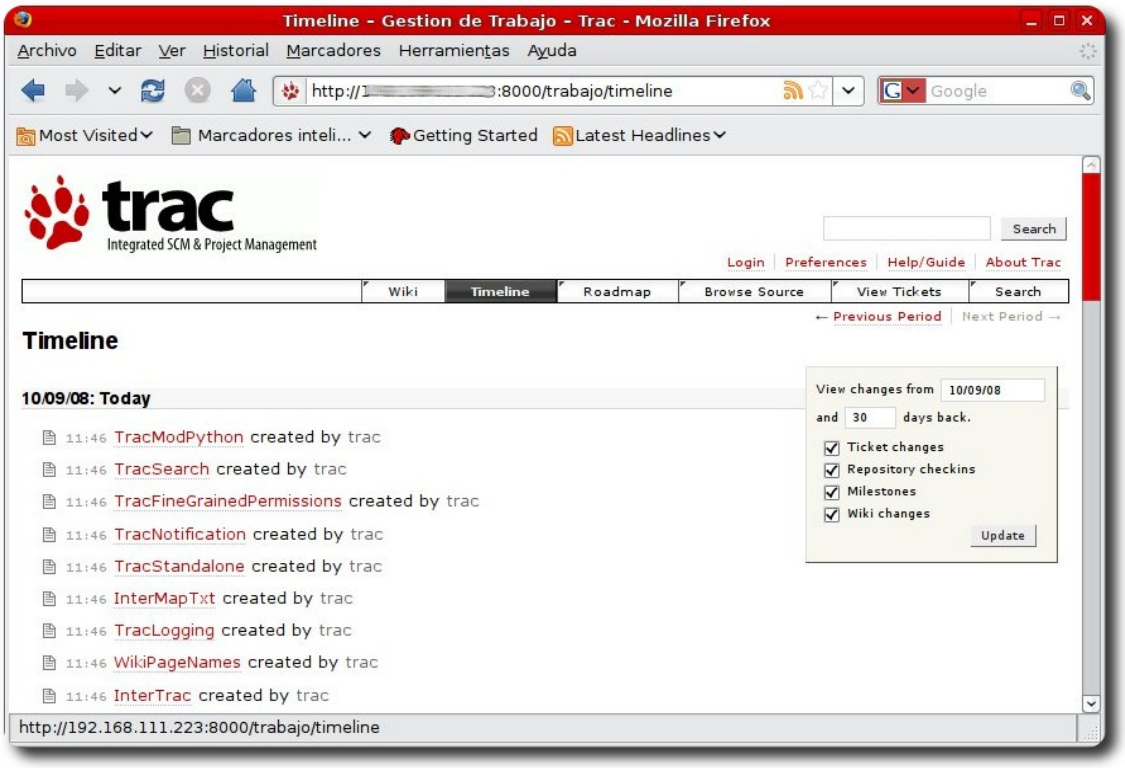


Gráfico 6: Timeline

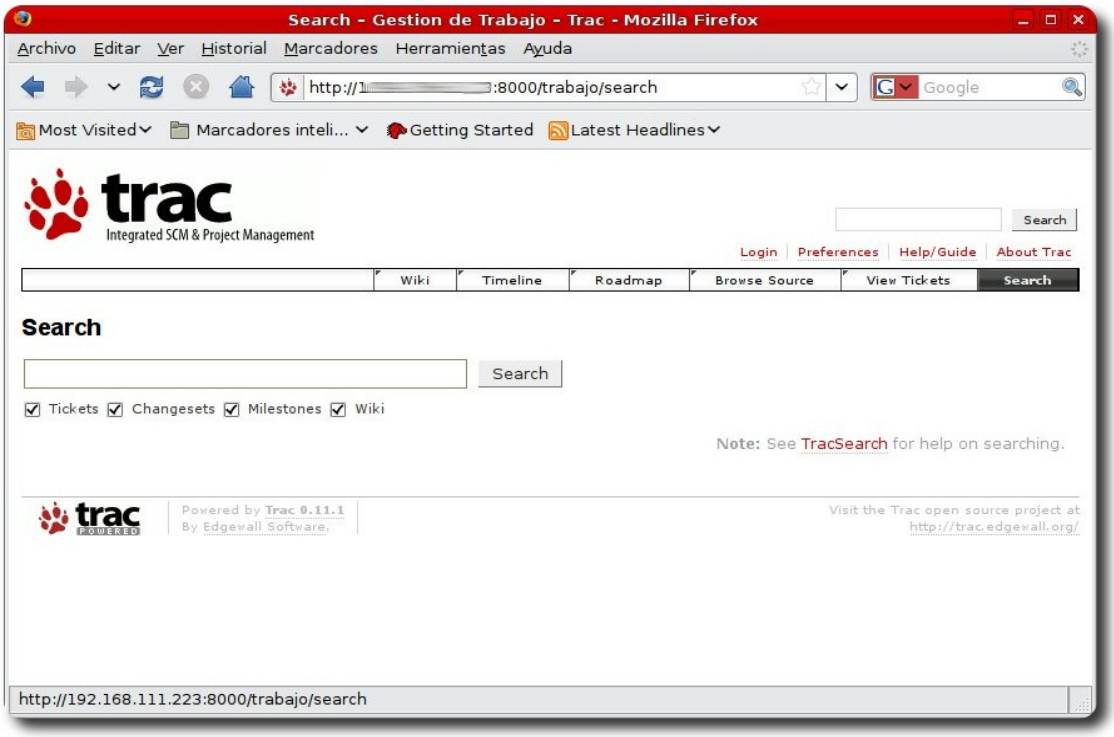


Gráfico 7: Búsquedas

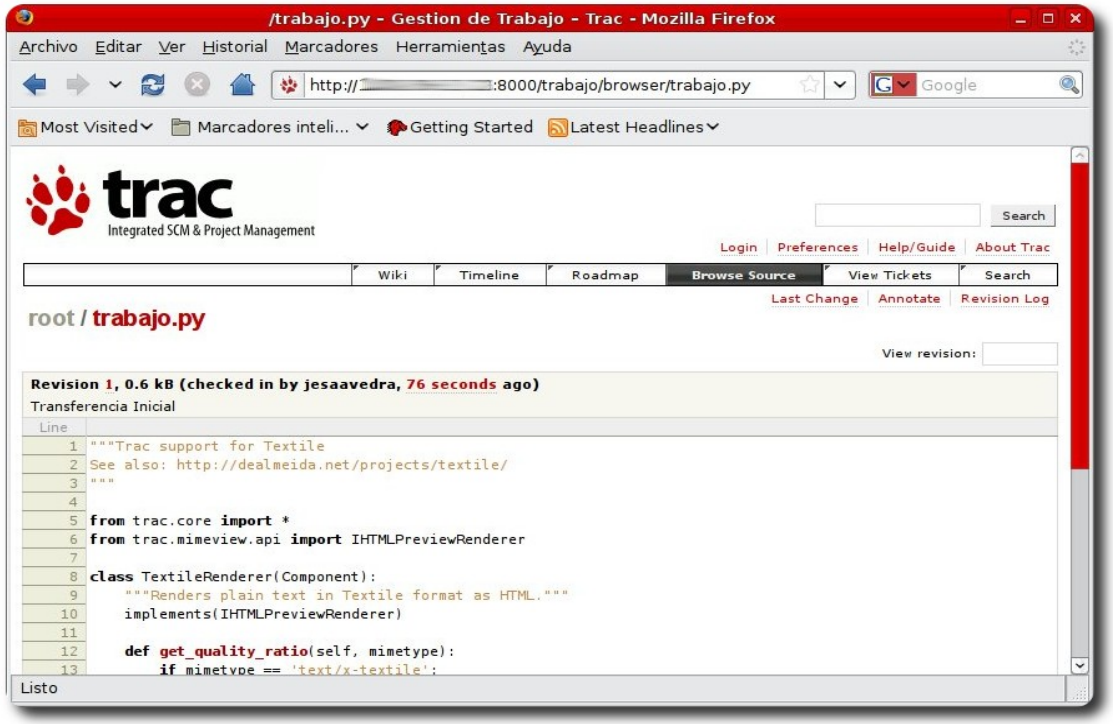


Gráfico 8: Acceso al control de versiones

Referencias

[1] Proyecto **Trac**: http://trac.edgewall.org/

[2] Componentes adicionales: http://trac-hacks.org/

[3] Acceso a repositorios libres: https://opensvn.csie.org/

Autor

Esteban Saavedra López

Líder de la Comunidad ATIX (Oruro – Bolivia)
Activista de Software Libre en Bolivia
jesaavedra@opentelematics.org
http://jesaavedra.opentelematics.org

Experiencia de desarrollo sistema SAFU

(Sistema Académico Financiero Universitario)

En el artículo vamos a ver como fué el desarrollo inicial de SAFU "**Sistema Académico Financiero Universitario**". Sin embargo queremos presentar dos perspectivas, es decir vamos a ver el proyecto desde el ángulo social y desde el ángulo técnico, Principalmente por que el proyecto ha tenido una connotación especial, ya que se adoptaron nuevas políticas de desarrollo que para casi todos los miembros del equipo fueron nuevas o relativamente nuevas.

Introducción

Queremos mostrar la experiencia en el desarrollo del proyecto **SAFU** (Sistema Académico Financiero Universitario), proyecto con el que ha ido evolucionando todo el equipo de trabajo, conociendo e identificándose con políticas e ideologías del software libre, siendo esta la raíz del cambio dentro la Universidad de Aquino Bolivia.

La evolución del proyecto no solo se refleja en el proyecto SAFU, sino que se extiende hasta el trabajo particular de cada uno de los miembros del equipo, que ahora somos impulsores y participes de la comunidad de Software Libre de la UDABOL, cada equipo se hace cargo del desarrollo no solo a nivel de Software, sino de capacitación basados en el uso de Sistemas Libres, promovemos la utilización de sus herramientas, impulsamos los sistemas GNU/Linux, organizando diversos, talleres, cursos, exposiciones y congresos.

En el poco tiempo que el equipo viene trabajando dentro la universidad a tenido avances significativos, realizando una migración total en cuanto a servidores de datos, aplicaciones y plataformas hacia el uso de Software Libre, logrando llevar la lógica empresarial y administrativa de la Universidad hacia una plataforma totalmente libre, segura y estable.

Mostraremos el aspecto técnico y también el aspecto social de la adopción de la filosofía del software libre dentro del equipo de desarrollo.

SAFU "El Desarrollo Como Experiencia"

La Universidad de Aquino Bolivia, cuenta con cuatro sedes en las ciudades de Cochabamba, La Paz, Oruro y Santa Cruz. En la sede de la ciudad de La Paz se había desarrollado un sistema académico el año 2000 el cual tenía como lenguaje de programación Delphi en la versión 6 y el motor de base de datos Microsoft SQLServer, este sistema estaba implementado a nivel nacional y todo el mantenimiento y desarrollo del mismo se hacia de manera centralizada por el Centro de procesamiento de Datos de la sede.

Debido al diseño interno de este sistema, su escalabilidad se hacia muy pesada y más aún el mantenimiento, llegando a tener versiones diferentes instaladas en cada sede. Por este motivo en las demás sedes se crearon pequeños departamentos de desarrollo de sistemas, los cuales producían sistemas informáticos sobre la base de datos intentando parchar las deficiencias o falencias del sistema base.

Estos programas eran incompatibles, se tenía una forma de trabajo particular o muy regionalizada, de la misma forma la organización y procedimientos eran específicas para cada caso creando escenarios diferentes en cada sede.

Cada sede tenía diferentes herramientas de desarrollo (lenguajes de programación, gestores de bases de datos), el mayor problema eran los pequeños sub-sistemas que funcionaban específicamente en otras bases de datos por lo que la integración nacional se hacia muy difícil.

Esta forma de trabajo ocasionó en muchas ocasiones la duplicación de esfuerzos para la resolución de problemas comunes. Es decir el escenario previo era una auténtica torre de Babel, debido a la falta de coordinación entre las diferentes sedes.

Debido a ésto, surge bajo estas circunstancias la necesidad de tener un proyecto que haga un análisis profundo de los requerimientos nacionales, y plantee el desarrollo de un nuevo sistema unificado y con metas únicas. Que unifique los esfuerzos nacionales para poder crear una herramienta que solucione los problemas de manera global.

El Inicio del proyecto

El departamento de desarrollo de Sistemas de información fue creado el año 2004 en la sede La Paz con la intención de crear un nuevo sistema para la universidad, pero este proyecto fue retrasado debido a que se puso prioridad para atender otros requerimientos que tenía la universidad. Se tuvo una experiencia de trabajo sobre el sistema SAGU (**`http://sagu.solis.coop.br/`**) en su versión 1, pero no se tenía el equipo necesario para afrontar un proyecto de esta dimensión. El trabajo realizado en SAGU fue la traducción y análisis de los procesos internos del sistema, gran parte del trabajo del sistema SAFU está basado en SAGU.

Por ésto, a finales de 2005 e inicios de 2006 se decide crear un departamento nacional de desarrollo para poder afrontar un proyecto más grande y que pueda implementarse a nivel nacional. Se realiza una integración nacional de los departamentos de desarrollo para conformar un único equipo de trabajo, cumpliendo objetivos comunes al servicio de la Universidad desde una perspectiva de trabajo nacional.

El equipo se organizó en jefes de desarrollo para cada sede y un jefe nacional de desarrollo, quien sería el directo responsable de los diferentes equipos. Se adoptaron normas y políticas que hagan posible el trabajo dentro la universidad, todas ellas basadas en la experiencia de proyectos de desarrollo bajo la filosofía del software libre.

Código Abierto y Normativas de Codificación

Se incursionó dentro de normas y políticas de desarrollo tomando como base las normas de desarrollo de Doxigen y Debian. Hasta éste momento el código fuente de los desarrolladores era propio de cada persona, sin embargo no era posible continuar de ésta forma, al encarar un proyecto mayor, el código fuente debía ser abierto y principalmente comprensible para el trabajo de cada uno de los integrantes del equipo, por lo que todos los integrantes tuvimos que mentalizarnos en una forma de trabajo común, tratando de ser explícitos y sobre todo ordenados, ésto era vital ya que todo el código y los módulos se centralizarían para funcionar como un solo sistema.

En este punto es importante destacar la influencia que tiene la reglamentación de código y las normas para codificar. Cualquiera puede escribir código fuente para si mismo, pero se necesita disciplina y constancia para poder escribir un código limpio y comprensible; se dice que un código es comprensible, cuando las personas que no han escrito tal código, pueden entender fácilmente que es lo que dicho código hace.

Uno de los aspectos un tanto más complicados de la integración de los equipos fue precisamente normalizar los diferentes estilos de codificación, hacer que entre todos no solo tengamos el mismo lenguaje, sino que también podamos comunicarnos de forma fluida por medio del código que se producía. Como se había mencionado el

código de los diferentes sistemas era algo muy personal, razón por la cual también se puede decir que era muy desordenado y tosco, De hecho ahora vemos hacia atrás y descubrimos muchos defectos que antes no se percibían. Al adoptar la forma de trabajo distribuida y de código abierto, aprendimos no solamente una nueva ideología, nos auto educamos para hacer las cosas con mayor cuidado, de forma que todas las personas involucradas entiendan lo que hacemos.

Fue una gran experiencia descubrir diferentes estilos de programación unos mejores que otros, sin embargo, todos fuimos mejorando la forma como desarrollábamos los sistemas al estudiar y aprender la forma como otros lo hacen. Entendimos que una forma importante de aprender a codificar es leer el código que hace otra persona, de la misma forma aprendimos que muchas personas trabajando sobre el mismo código es una forma de optimización de código altamente efectiva.

El Framework de trabajo FRADERA

A medida que se realizaba el análisis inicial del sistema SAFU se vió necesario proveer a los programadores de una base en la cual no tengan que realizar dos veces el mismo trabajo. Para esto se analizó la utilización de un framework que dé la base de programación, pero resultaba muy alta la curva de aprendizaje para los programadores. Por ésto se decidió realizar un framewok propio al cual denominamos FRADERA(Framework de desarrollo rápido).

El framework surgió como una pequeña colección de funciones comunes que todos utilizábamos, pero rápidamente se fue constituyendo en la base del proyecto, permitiéndonos la misma abstraernos de la lógica de programación elemental y concentrarnos en el desarrollo de alto nivel.

El ciclo de desarrollo del Framework es variado, tenemos una versión estable, sobre la cual trabajan los diferentes equipos, sin embargo cuando se precisa una utilidad común para los diferentes módulos de desarrollo, uno de los equipos o algún miembro del equipo trabaja en el desarrollo de dicho elemento necesario en el framework hasta que éste queda estable, cuando se lo logra, se pone en consideración de los equipos para que se realice pruebas y se verifique la utilidad, si es que se encuentra algún problema, cualquier miembro del equipo puede solucionarlo, o en su defecto se asigna el trabajo a algún miembro del equipo, cuando el problema queda solucionado, se considera nuevamente al framework estable y se continua con el desarrollo de los diferentes módulos.

De esta forma el framework, nos ayuda a tener un desarrollo uniforme, pues muchas de las tareas repetitivas ya están definidas por el mismo framework en forma de colección de clases, a las que solo hay que acceder para realizar una tarea común, por ejemplo la conexión a una base de datos, la autenticación de un usuario en un sistema, entre muchas otras, es decir al dejar de lado este tipo de tareas hacemos que el desarrollo de los diferentes módulos del sistema sean uniformes, puesto que todos ellos tendrán cadenas de conexión similares, sistemas de autenticación similar, en resumen un comportamiento homogéneo, entonces las diferencias estarán definidas solamente por las necesidades especificas de cada modulo.

La utilización de FRADERA, no fue bien recibida en primera instancia dentro de los equipos de trabajo, principalmente por que fue un framework nuevo y poco estable. Inicialmente solo era una colección de utilidades y obviamente aún tenía muchas cosas que no estaban presentes dentro del framework como tal. Es en éste punto donde se siente cierta resistencia al cambio. Una cosa era compartir tu código y hacer que éste sea comprensible. Sin embargo utilizar y hacerse del código y la lógica de programación de otros es más difícil, y es también donde se encontró mayor resistencia.

Los diferentes miembros del equipo inicialmente mostraban un constante rechazo al código de FRADERA argumentando la inestabilidad, o la carencia de elementos y ésta fue la barrera más difícil de superar, el equipo se dedicaba a descubrir continuamente errores en FRADERA, sin

darse cuenta que todo el equipo debería estar también dentro de este proyecto. Recordemos que una de las libertades básicas del Software Libre es la libertad de modificar el código para el bien propio, y aún cuando todos teníamos las herramientas necesarias para poder hacer este cambio, algunos de los equipos asumían un papel pasivo de forma que todos los problemas debían ser resuelto por los encargados de FRADERA. Este punto empezó a tornarse crítico pues por un lado teníamos el desarrollo de SAFU como sistema y por otro lado se tenía el desarrollo de FRADERA. La falta de comunicación hacia inestable el desarrollo, principalmente por que unos no satisfacían las necesidades de otros y por que los otros no tenían el apoyo suficiente.

Nuevamente se atravesó por el problema de tener trabajo duplicado, y desordenado, en aquel tiempo podíamos ver dentro de SAFU porciones de código que estaba definida por el framework y porciones de código independiente por los desarrolladores, teniendo de ésta forma un desarrollo inconstante y no necesariamente comprensible. Los diferentes desarrolladores empezaban a distorsionar más el código, creando cada cual sus propias librerías y funciones básicas que en muchos casos se encontraban presentes ya en FRADERA. ésto solo ocasionaba nuevamente una duplicación de trabajo y mayor consumo de recursos, además de un código desordenado y confuso. Al punto que existieron módulos que tuvieron que rehacerse por completo.

Comunicación

Esta se convirtió en el problema fundamental por el cual había inestabilidad en el equipo. Exactamente, uno de los pilares del Software Libre es la “Comunicación”, en esta etapa del proyecto nuevamente habían surgido las individualidades dentro los equipos de desarrollo, cada uno de los equipos de desarrollo quería ser más importante que los otros.

En este punto se realizó el ordenamiento del departamento nacional bajo reglas estrictas para no afectar al proyecto. Dentro de la lógica de los programadores del mercado boliviano es difícil poder mostrar que compartir es mejor que competir. De acuerdo a la formación que se da en la universidades nunca se les dice que compartir es bueno, al contrario se cierra en una lógica de egoísmo.

Herramientas de Comunicación

Inicialmente se utilizó un foro de discusión, sobre el cual se intercambiaban opiniones, sin embargo con el tiempo ésto se volvió muy tedioso. Y otros medios como listas tampoco resultaban muy fluidos. Se eligió comunicación directa por medio de clientes de mensajería instantánea, éste era el medio más rápido y fiable así que ahora mantenemos una comunicación fluida y cada cierto tiempo celebramos reuniones virtuales sobre las que discutimos diferentes puntos de nuestro trabajo, de tal forma que el desarrollo del proyecto esté correctamente organizado y los errores y problemas sean comentados y discutidos oportunamente sin dejar que dichos problemas crezcan más de lo debido. Esto mejoro mucho la comunicación entre los diferentes equipos.

El wiki es una herramienta importantísima para el desarrollo del proyecto ya que por este medio hemos construído la documentación de todo el proyecto, de hecho este artículo también se construyó de la misma manera.

También utilizamos el blog, para registrar nuestros avances y realizar alguna consulta de forma, para que quien pueda colaborar en algún modulo, o a quien le vaya a ser útil algo que se ha desarrollado sobre el sistema sea de utilidad paro todos los equipos.

Cabe destacar que inicialmente la comunicación era casi individual, es decir entre los diferentes miembros de los equipos, sin embargo en la actualidad la comunicación es general, las reuniones se hicieron más fluidas y por tanto la forma misma del desarrollo empezó a mejorar. Quedó claro que si un elemento falla todo el equipo falla, por lo que ahora el apoyo es mutuo y mejor, razón por la que en este momento tenemos un mejor avance y una mejor pre disponibilidad de trabajo.

Control de calidad y trazo de errores

Otra herramienta importante dentro del desarrollo cooperativo y el desarrollo distribuido son las herramientas de control de calidad y trazo de errores, al ser un grupo grande de personas no bastaba solamente con decir cuales eran los problemas que habíamos encontrado, sino también registrarlos para poder hacer un oportuno seguimiento del mismo.

De ésta forma además de los propios desarrolladores, nos convertimos también en los primeros testers del sistema, donde aprendimos la importancia que tiene el reporte de errores y la correcta forma de hacer un reporte de error, muchas personas dentro del Software Libre han tenido la oportunidad de reportar un error, sin embargo la mayoría de estas personas no lo hace correctamente, de hecho nosotros mismos no lo hacíamos correctamente, porque es importante mencionar que reportar un error no es solamente decir ésto no funciona, sino el verdadero reporte consiste en demostrar la falla y ver si ésta es de alta o baja incidencia para que el desarrollador pueda darle mayor o menor prioridad, de acuerdo a los requerimientos del sistema.

Se dice que si el 30% de los errores que fueron registrados para un software tuvieran información relevante, éste estaría muy cercano a la perfección.

Desarrollo Distribuido y Control de versiones

El desarrollo del proyecto es totalmente distribuido, en las diferentes ciudades por lo que necesitábamos alguna herramienta que nos permita, tener un control de los archivos que se iban modificando, y esta herramienta fue Subversión, inicialmente acostumbrarnos a ella fue un tanto complicada, en la actualidad no podemos concebir el desarrollo del proyecto sin dicha herramienta, nos dimos cuenta que Subversión es también la herramienta fundamental de proyectos aún más grandes que el nuestro, no existe forma alguna de sincronizar los elementos de desarrollo sin la utilización de un sistema de manejo concurrente de archivos.

Al principio también tuvimos problemas con subversión, teníamos archivos que eran sobre escritos, y otros modificados sin intención e incluso hasta archivos borrados, sin embargo nuevamente fueron problemas de comunicación por lo que también para esta etapa de desarrollo adoptamos métodos de trabajo y normas, los cuales nos ayudaron a una mejor organización; por ejemplo actualmente dentro de nuestras políticas está tratar de hacer una revisión extensiva de los archivos modificados en forma local, una vez que estos archivos hayan atravesado por algunas pruebas primero ejecutar un update antes de un commit (subir nuestro archivo modificado). Gracias a que subversión es un sistema sólido y estable logramos resolver muchos de los problemas que teníamos inicialmente y ahora el desarrollo es rápido y fluido, las actualizaciones son bastante rápidas y la detección de errores son igual de rápidas.

La parte técnica

Arquitectura de diseño sistema SAFU

Diseño conceptual.

Para el desarrollo del nuevo sistema **SAFU**(Sistema Académico Financiero Universitario) se utilizó la metodología de diseño RUP para el diseño conceptual del mismo. En la fase de diseño se partió del concepto que el nuevo sistema debía separarse en tres módulos principales:

- ✔ Módulo Académico.
- ✔ Módulo Financiero.
- ✔ Módulo de Recursos Humanos.

El patrón experto da al sistema la capacidad de que cada módulo y submódulos que lo componen sean expertos solo para lo que fueron diseñados, por ejemplo el módulo de facturación solo realiza facturación, y no toca otras áreas que no le corresponden. El bajo

acoplamiento permite que cada módulo sea especialista en su área y sea casi independiente en su funcionamiento. Siguiendo este patrón de diseño cada uno de los módulos podría funcionar sin necesidad de los otros. El patrón de alta cohesión sirve para que cada módulo y cada uno de los submódulos que lo componen puedan tener un flujo de trabajo interno sin crear dependencias muy fuertes con otros módulos o submódulos. Al seguir estos patrones en el diseño conceptual y el diseño de programación del sistema permite tener una alta re utilización de las herramientas desarrolladas.

Adicionalmente para la fase de diseño conceptual del sistema se crearon equipos interdisciplinarios de trabajo divididos en dos áreas de trabajo. Un equipo conceptual y un equipo técnico ambos liderados por una cabeza que es el analista de sistemas. El equipo conceptual está conformado por personal de la universidad que trabaja y conoce el flujo de información de su área de trabajo. El equipo técnico conformado por programadores es el encargado de realizar el desarrollo de los módulos a nivel de interfase , el sistema según los lineamientos que coordinó el analista con el equipo conceptual. Cada analista de sistemas trabajó sobre una lista de requerimientos previamente desarrollada con las autoridades de la universidad, en base a la cual diseñó la arquitectura base del módulo asignado y la refino en reuniones de coordinación con el equipo conceptual. También se le designó la tarea de realizar el control de avance del equipo técnico en la fase de desarrollo.

Diseño técnico.

En la parte técnica se definió el uso de herramientas libres para el desarrollo del nuevo sistema. Esta elección se basó en la experiencia de desarrollo del departamento con sistemas anteriores.

Las herramientas definidas para el desarrollo son:

- ✔ Lenguajes de programación
  - ✔ PHP en su versión 4 por ser la estable en la fase inicial del proyecto.
- ✔ Ajax para dar mayor versatilidad a la interfase de usuario.
- ✔ Herramientas de desarrollo
  - ✔ FRADERA, framework de desarrollo rápido, el cual se desarrollo a la par que se avanzaba el proyecto.
- ✔ Gestor de base de datos
  - ✔ PostgresSQL dada su robustez y soporte
- ✔ Servidores de implementación
  - ✔ Sistema operativo GNU/Linux con la distribución DEBIAN
- ✔ Apache 2.0

En base a las herramientas seleccionadas y debido a las características que tiene PHP se definió usar el patrón de diseño MVC (modelo, vista, controlador), orientando a objetos casi todo el esquema de programación hasta donde permite hacerlo PHP. Para poder utilizar el patrón de diseño MVC se creo el framework FRADERA que nos permite hacer la separación de la arquitectura de la aplicación en varias capas. Y que nos da las siguientes ventajas:

- ✔ Alta reutilización de componentes que son usados comúnmente en el desarrollo de aplicaciones.
- ✔ Limitación de aparición del número de errores en las actividades reiterativas evitando la duplicación de código.
- ✔ También permita la reutilización del conocimiento ya aplicado en solución de diversos de problemas.
- ✔ Orden y sistematización de la aplicación y futuras aplicaciones. Ya que se define una estructura de directorios y de archivos de configuración.
- ✔ Utilización de estándares de codificación para el equipo de programadores.
- ✔ Facilidad de expansión de nuevas funcionalidades a nivel del framework y de las aplicaciones desarrolladas

bajo el mismo.

Con el modelo MVC se tiene las siguientes ventajas:

- ✔ Modelo, la capa de objetos de negocio y del acceso a los datos se crean desde el framework. Es decir si analizó el código de la aplicación no puedo ver consultas directas a la base de datos, sino pase de parámetros que arman las consultas dinámicamente el momento de ejecutarlos.
- ✔ Vista, define de qué modo representar los objetos de la capa modelo y los elementos del interfaz del usuario. Es responsable del formateo apropiado de los datos del modelo en función del método de la presentación (por ejemplo, el navegador web o PDF),
- ✔ Controlador, es el "director" que, en respuesta a las acciones del usuario, descarga los objetos desde el modelo y los entrega a una vista seleccionada.

Con este diseño se permite la separación de los datos y los reglas que rigen para su presentación a los usuarios finales.

Como diseño interno del framework se conceptualizó niveles bajo el siguiente esquema:

- ✔ FRADERA
  - ✔ Nivel de aplicación
    - ✔ Módulos específicos
      - ✔ Submódulos
        - ✔ Tareas

Bajo este diseño el mantenimiento de cada uno de los módulos es más sencillo. Esto también permite poder controlar los accesos de los usuarios en cada uno de los niveles. También permite poder tener un control total sobre las operaciones realizadas en el sistema bajo el sistema de huellas de operación(logs) que maneja el framework y que puede hacerse extensible sin necesidad de reescribir todos los módulos solamente la parte de seguridad del framework.

El diseño de seguridad de acceso al sistema está fuertemente ligado al esquema presentado anteriormente, por éste motivo se puede dar permisos a cada usuario desde el ingreso a una determinada aplicación y descendiendo por sus módulos y submódulos hasta llegar a las tareas específicas que pueden ser configuradas dentro del módulo de administración que tiene el framework.

Para poner un ejemplo práctico podemos mencionar, una aplicación seria el módulo académico al cual se puede dar acceso a un usuario, luego se debe dar los permisos para el uso de los diferentes módulos de la parte académica, luego se debe dar los permisos de uso de cada uno de los submódulos por ejemplo el submódulo de consultas. Ya dentro de los submódulos se puede dar permisos para las tareas específicas que pueden ser tareas comunes de altas, bajas y modificaciones o tareas definidas por la propia aplicación, por ejemplo impresión de actas que se puede tomar como una tarea dentro del submódulo de consultas.

Para el diseño interno de la estructura de datos se decidió utilizar una sola base de datos que está divida también en las tres áreas de trabajo. Para poder tener un diseño flexible a nivel de estructura de datos se utilizó normas de codificación para los nombres usados dentro de este diseño. Tratando de llegar un nivel de normalización de base de datos alto. Con este diseño se crea la restricción de que los datos de más bajo nivel(transacciones básicas) no puedan modificarse desde el gestor de administración del DBMS con facilidad. Y obliga a los administradores del sistema a tener que utilizar la interfase del sistema para poder realizar modificaciones en estos datos. Estas operaciones de modificación de datos serán registradas en el sistema de huellas de operación del framework permitiendo tener un nivel alto de auditoría sobre las mismas. Adicionalmente sobre estas medidas de seguridad se debe plantear procedimientos de administración de las bases de datos que permitan evitar accesos no deseados para mantener la integridad de los datos del sistema.

Con todo el esquema para el diseño conceptual y técnico del sistema se da la facilidad de crear niveles de seguridad no

dependientes entre si. Se pueden definir políticas de seguridad a nivel de implementación en los servidores de aplicaciones para definir los accesos a las diferentes áreas del sistema, definir políticas para el acceso a los servidores de base de datos. Con ésto se puede definir niveles de auditoría ligados entre si a los diferentes responsables de cada servidor y de el sistema en si.

Conclusiones

Como verán el pensamiento y la utilización de herramientas de software Libre nos ha dejado mucha enseñanza, la cual seguimos cultivando, en la actualidad una gran parte de nosotros estamos íntimamente relacionados con el Software Libre, ya que tuvimos la oportunidad de experimentar y trabajar a fondo con el mismo por lo que podemos decir las ventajas que tiene la utilización de estas herramientas y la gran calidad de los proyectos relacionados con el Software Libre.

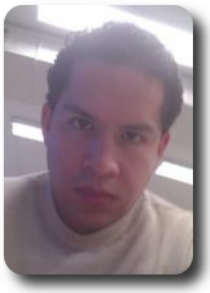
Es importante destacar que así como nosotros fuimos aprendiendo con la experiencia de desarrollo, nuestra tarea se ha ampliado a tratar de difundir el Software Libre como herramientas de trabajo en la Universidad en la cual trabajamos, de hecho en nuestro país la UDABOL es la institución que a mostrado un apoyo mayor hacia el uso de tecnologías libres y éste es un gran logro principalmente considerando que es una universidad privada, los miembros del equipo de desarrollo tratamos siempre de compartir nuestra experiencia con la utilización de sistemas libres, dentro de nuestras actividades en la universidad nos dedicamos también a proponer charlas, talleres, encuentros, congresos para impulsar el uso del Software Libre, de alguna manera dentro la comunidad de software Libre Bolivia algunos de los miembros del equipo tenemos cierta representación a nivel Nacional.

Hemos mostrado que la UDABOL está realizando una migración completa hacia plataformas libres, no solo a nivel administrativo, sino que también inicia un camino de utilización de herramientas libres para la educación de los estudiantes. Prueba de ello también es la estructura de los portales de la universidad que utilizan Software Libre “Joomla” de la misma forma contamos con una plataforma de Educación a Distancia “Moodle” y varios otros sistemas como el control de biblioteca y el sistema de clínica odontológica todas ellos basados en Software Libre.

Entre los proyectos que tenemos está la realización de cursos de capacitación, no solo para los estudiantes sino también para los docentes de la institución. de forma que se vaya realizando una completa migración hacia Software Libre.

Autores

Esteban Lima Torricos
elima@udabol.edu.bo



Vladimir Castro Salas
vladimircs@gmail.com

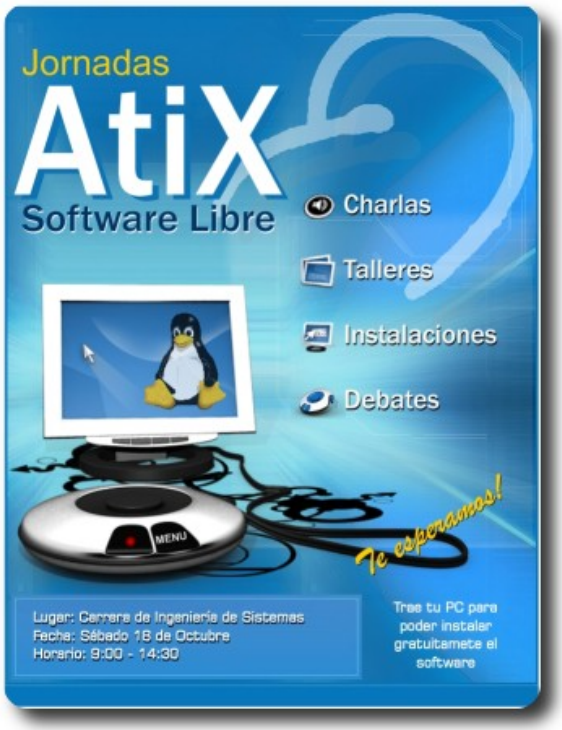
ATIX

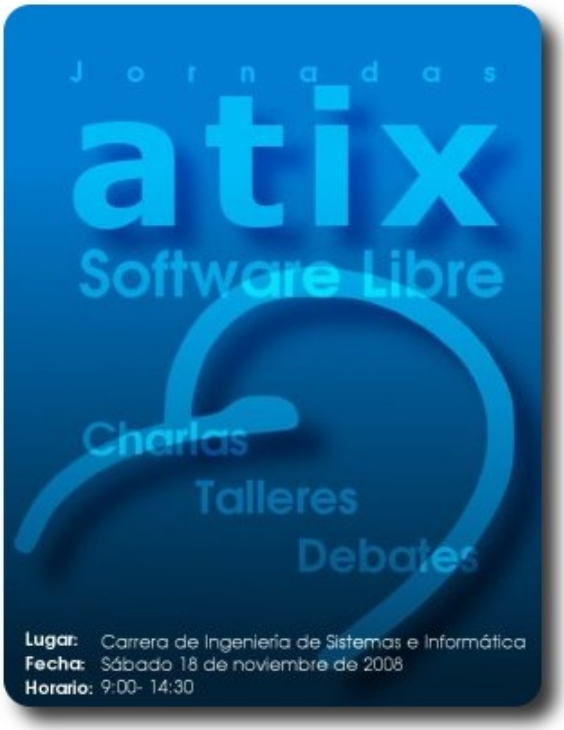
Noticias

Jornadas de Software Libre Atix

Hasta la fecha son varias las actividades que la comunidad de software libre Atix, ha organizado, entre charlas, talleres, festivales de instalación, etc; pero a partir de la fecha los miembros de la comunidad han decidido numerar las actividades en base a un nombre común, es por eso que a partir de ahora, nuestras actividades serán "**Jornadas de Software Libre Atix**", denotando la primera o la n-sima según corresponda, esta actividad será la que englobe nuestras futuras actividades.

Como empezamos una nueva etapa en la comunidad, esta se caracterizó por tener 2 afiches que sirvieron para publicitar ésta actividad.





Jornadas de Software Libre Atix

Algunos de los miembros de la comunidad Atix y algunos asistentes a las jornadas

Miguel explicando la necesidad de contar con una ley en pro del software libre

Howard, mostrando el poder del Framework Cake

Cristhian mostrando la variedad de clientes de mensajería instantánea sobre GNU/linux

Arnold demostrando el poder de Latex

Momento de los debates

Asistentes siguiendo los talleres

En esta oportunidad estrenamos nuestros polos, con el distintivo de la comunidad

InfoNews – Doble Clic

Usuarios de Internet pierden el interés por Chrome

Luego de un gran interés por el nuevo navegador de Google que en inicio era noticia en todos los foros y blogs, google chrome ha caído al mínimo, con 0.77% según la empresa de análisis y estadísticas en línea Net Applications, una de las razones podría ser la débil publicidad que Google ha dado al producto. En efecto, el navegador sólo es publicitado mediante la propia plataforma publicitaria de Google, Adwords.

Nokia estrenará el rival de Iphone

Nokia ha elegido España como primer país del mundo en el que comercializará su nuevo y esperado teléfono, el 5800 XpressMusic, llamado a competir con el iPhone de Apple, por supuesto, con pantalla táctil.

La música será uno de los puntos fuertes del Nuevo 5800 XpressMusic,el software Symbian S60 (específicamente creado para dispositivos móviles) y la navegación por Internet con conexiones 3G, WiFi y HSDPA sus baluartes, además de una Cámara de fotos y vídeos de 3,2 megapíxeles, GPS y Nokia Maps con navegación guiada por voz son otras de las funcionalidades que incorpora este teléfono.

SIMO estrena portal y red social

La web de la feria se convierte en un espacio para que expositores y consumidores puedan hacer transacciones comerciales todo el año

La Feria Internacional de Informática, Multimedia y Comunicaciones, que se celebrará en IFEMA del 11 y al 16 de noviembre feria de Madrid, está casi lista para convertirse en el escaparate del sector TIC a nivel internacional durante cinco días, pero antes de abrir sus puertas, innova ya a través de Internet, que servirá de espacio comercial donde los expositores podrán presentar y promocionar sus productos, será como un “Almacén Tecnológico virtual”.

Ballmer admite que Google Apps está haciendo efecto dentro de MS Office

En una discusión entre Steve Ballmer de Microsoft y dos analistas de Gartner, Ballmer admite que Google Apps está disfrutando de una ventaja sobre oficina por los usuarios que quieren compartir sus documentos. Él señala que “Google tiene el plomo, pero, si somos buenos en la publicidad, competiremos con ellos en el negocio de consumidor.” Independientemente de si son buenos en la publicidad todavía está en la pregunta, si sus tentativas recientes tienen alguna indicación. Ballmer también hizo las declaraciones que indicaban una cierta clase de arreglo con Yahoo!. En relación con Windows Vista, él dijo que Microsoft estaría "listo" a la hora de lanzar Windows 7.

Autor

Ivonne Karina Menacho Mollo
Titulada de la carrera de Ingeniería de Sistemas e Informática (F.N.I.)
Conductora programa radial “**Doble Clic**”
ivonnekarina2003@hotmail.com

8vo congreso nacional de software libre

La Paz, Bolivia
13, 14 y 15 de
noviembre de 2008

http://congreso.softwarelibre.org.bo

organizan:

- Comunidad Software Libre Bolivia
  www.softwarelibre.org.bo
- www.boliviaOS.org
- www.VocesBolivianas.org
- www.SolMujeres.org
- www.UbuntuBolivia.org
- www.Runasimipi.org
- Revista Atix

temática:

- 2a. jornada de gestión pública y software libre
- software libre y empresa
- software libre y educación
- distribución BoliviaOS
- 2o. encuentro ubuntu Bolivia
- 1er. encuentro SolMujeres

conferencias, talleres, mesas redondas,
barcamp, demostraciones técnicas...

contactos:

congreso@softwarelibre.org.bo
772 95 196 (Esteban)
765 67 555 (Boris)
720 29997 (Daniel)

inscripción:

- profesionales: Bs. 100
- estudiantes:
  de La Paz/El Alto, Bs. 50
  de otros lugares, Bs. 30

jueves 13 y viernes 14:

Universidad Católica Boliviana "San Pablo"
Av. 14 de Septiembre, esq. calle 2, Obrajes
Sala de Docentes, 5o. piso, Bloque D

sábado 15:

Auditorio Entel
Federico Zuazo No. 1771
La Paz

auspician: entel
www.entel.bo

Carrera de Ingeniería de Sistemas
www.ucb.edu.bo

www.adsib.gob.bo

www.opentelematics.org

www.openit.com.bo

Comics

Nuestro CIO, esta encriptado por razones de seguridad.

Los términos parecidos pueden confundirse en entornos no técnicos

.

Conociendo lo Nuestro

Chuquisaca

Vista de la Catedral en la plaza principal de Sucre

Teatro Gran Mariscal de Ayacucho

Portada de la casa de la Libertad

Castillo de la Glorieta

Vista de patio de la Casa de la Libertad

Torre de la Catedral

Libres para pensar, libres para decidir, libres para crear

Arte Libre

Te ofrecemos este espacio para mostrar tu Creatividad

Envíanos tus diseños y creaciones para publicarlos

ARTE EN LIBERTAD

Comencemos por la pregunta que posiblemente muchos se hacen a sí mismos cuando se sientan en sus Ordenadores: Si quiero dibujar, ¿Cuánto me va a costar la Licencia?, no sería mejor preguntarnos ¿Hoy con que me siento inspirado?.

Desde hace algún tiempo ya están disponibles herramientas libres de Diseño que rompen con las barreras económicas de muchos usuarios fieles al arte y que demuestran que ser costosos no es sinónimo de alta calidad. El mundo del Arte digital ha crecido exponencialmente gracias al apoyo de comunidades de artistas y desarrolladores que juntos se han propuesto crear herramientas sencillas y prácticas que cumplan con el objetivo más básico del dibujo... Crear. Conozcamos tres de esas herramientas:

GIMP - Es una aplicación para la manipulación de imágenes que convierte el área de trabajo en un divertido conjunto de ventanas configurables. Parte de la idea de que menos es más, y deja que el usuario decida que herramientas tener en su interfaz principal. Con GIMP puedes editar desde gráficos simples hasta realizar montajes y retoques fotográficos, como también crear maquetado para imprentas, páginas web y mucho más.

INKSCAPE - Nos permite realizar gráficos que nunca se pixelan. Esto se logra mediante el uso del formato SVG, que no es más que un principio de imágenes vectorizadas. Con esta herramienta, generar gigantografías y ajustar gráficos a nuevas modificaciones para animaciones es más sencillo.

BLENDER - En el mundo del Arte libre también existe espacio para los gráficos en 3 dimensiones y esta es la mejor herramienta para realizarlos hoy en día. Desde un uso simple para planos o incluso para montajes en Cine (Ej. Spiderman) encuentran su foco en Blender. Esta particular herramienta,

María "tatica" Leandro
www.tatica.org
Venezuela

Consultas

Contacto

Para solicitar cualquier información, puedes contactar a:

- Esteban Saavedra López (jesaavedra@opentelematics.org)
- Williams Chorolque Choque (williamsis@gmail.com)

Publicación

Te invitamos a ser parte de la **Revista ATIX**. La forma de participar puede ser enviándonos:

- Artículos referidos a áreas como:
  - Instalación y personalización de Aplicaciones
  - Scripting
  - Diseño gráfico
  - Programación y desarrollo de aplicaciones
  - Administración de servidores
  - Seguridad
  - y cualquier tema enmarcado dentro del uso de Software Libre
- Trucos y recetas.
- Noticias.
- Comics.
- Links de interés.

Marcamos Huella

http://atix.opentelematics.org