AÑO 0
NÚMERO 1
2012-12-03

#1

<Cobra>

HD Hackers & DEVELOPERS

Magazine digital de distribución mensual
sobre Software Libre, Hacking
y Programación

Staff

Celia Cintas	Licenciada en Informática
Eliana Caraballo	Ingeniera de Sistemas
Elizabeth Ramirez	Ingeniera Electrónica
Eugenia Bahit	Arquitecta GLAMP & Agile Coach
Indira Burga	Ingeniera de Sistemas
Laura Mora	Adm. de Redes y Sistemas
María Jose Montes	Tec. en Informática de Gestión
Milagros Infante	Est. Ingeniería de Sistemas
Sorey Garcia	Arquitecta de Software
Yecely Diaz	Maestra en Inteligencia Artificial

HD Hackers & DEVELOPERS #1

"... Hacker es alguien que disfruta jugando con la inteligencia.."

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

Acerca de

Hackers & Developers es un Magazine digital de distribución libre y gratuita, sobre Software Libre, hacking y programación.

Se distribuye mensualmente bajo una licencia Creative Commons.

Envía tu artículo

¿Quieres colaborar con HD Magazine? Puedes enviarnos tu artículo a **colabora@hdmagazine.org**

U! Dilo en público

¿Quieres enviar un comentario para que se publique en la zona **U!**? Escríbenos a:
lectores@hdmagazine.org
O publica tu mensaje en **Twitter** con el hashtag **#HDMagazine**

Contáctanos

¿Quieres enviarnos un comentario en Privado? Envíanos un e-mail a:
contacto@hdmagazine.org

Haz un donativo

¿Quieres apoyarnos económicamente? Envíanos un e-mail a:
donaciones@hdmagazine.org

Merchandising

Visita nuestro **Web Store y apóyanos económicamente** adquiriendo merchandising de Hackers & Developers
http://store.hdmagazine.org

Este mes en Hackers & Developers...

ASCII ART
Este mes: «Cobra»
by Anónimo

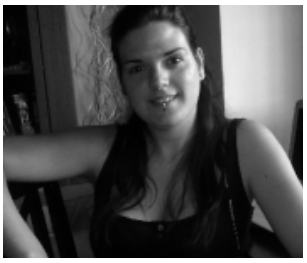
>> Pág. 48

Pásate a GNU/Linux con Arch Linux: Parte I

GNU/LINUX

Arch Linux: la distribución ideal para principiantes y expertos que quieran moldear su sistema a partir de una base mínima y... ¡Qué adoren la línea de comandos! En esta primera entrega, María José nos enseña como dar nuestros primeros pasos.

Escrito por: **María José Montes Díaz** (Archera & Python)



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

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

Redes sociales:
Twitter: **@MMontesDiaz**

Arch no se basa en ninguna otra distribución GNU/LINUX, pero se inspira en la simplicidad de Slackware, Crux y BSD. Fue creada por Judd Vinet en 2001 y su primer lanzamiento fue el 11 de marzo 2002. A partir de 2007, el proyecto es dirigido por Aaron Griffin.

Los cincos principios que constituyen la filosofía de Arch son:

1. **Simplicidad:** Se refiere a una estructura base sin agregados que permite al usuario moldear el sistema de acuerdo a sus necesidades.
2. **Precisión del código por encima de la comodidad:** Es decir, la elegancia y el minimalismo deberán permanecer siempre a la máxima prioridad.
3. **Centrado en el usuario:** Tiene como objeto ser cómoda y dándole el completo control y responsabilidad sobre el sistema.
4. **Abierto:** Utiliza herramientas simples, que son seleccionadas o construidas con filosofía de código abierto.
5. **Libre**: Proporciona la libertad de tomar cualquier decisión sobre el sistema.

"keep it short and simple!"

Una característica importante es que se trata de un *Rolling Release*. ¿Eso qué significa? Que no hay versiones, además, podemos actualizar el sistema con un sólo comando.

Instalación de Arch Linux: Parte 1

Lo primero que debemos hacer es descargar la última imagen de Arch en:

https://www.archlinux.org/download/

Con esta imagen podemos grabar un CD o DVD, usarla montada como imagen ISO, o pasarla a un dispositivo USB.

Para pasarla a un dispositivo USB, podemos utilizar el comando dd:

```
# dd if=archlinux.iso of=/dev/sdx
```

Siendo `/dev/sdx` nuestro dispositivo. Cuidado en este paso, pues la unidad destino perderá los datos. No se hace sobre una partición (`/dev/sdxy`), sino sobre el dispositivo completo. Antes de lanzar el comando, debemos asegurarnos que esté desmontado.

Al iniciar nos encontramos con un menú. Las dos primeras opciones son para iniciar la instalación (Boot ArchLinux). Una inicia Arch de 32 bits (i686); la siguiente inicia Arch para sistemas de 64 bits (x86_64). En caso de que nuestro equipo no soporte 64 bits, no aparecerá la opción.

Comenzamos:

1. Configuración del teclado.

El teclado por defecto viene configurado en inglés. Para ponerlo en Español:

```
# loadkeys es
```

2. Conectar la red

El demonio `dhcpd` se inicia automáticamente, con lo que, si nuestra conexión es por cable, debería estar configurada. En caso de que nuestro router no disponga de DHCP, deberemos asignar la IP de forma manual:

Nomenclatura: El nombre de la interfaz es, normalmente, **eth0**. Si hay más de una tarjeta de red, seguirán la secuencia, por ejemplo **eth1** , **eth2**...

Activamos la interfaz de red, le asignamos la dirección IP y añadimos la puerta de enlace (Gateway). Por ejemplo, nuestra interfaz es **eth0**, nuestra IP será 192.168.0.2 y nuestra puerta de enlace 192.168.0.1:

```
# ip link set eth0 up
# ip addr add 192.168.0.2/255.255.255.0 dev eth0
```

```
# ip route add default via 192.168.0.1
```

Nos falta definir los servidores de DNS. Para ello editamos el fichero `/etc/resolv.conf`

```
# nano /etc/resolv.conf
```

y los añadimos:

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```

En el caso de que nuestra conexión deba ser wifi (habrá que configurar después otra vez, puedes obviar este paso si no es necesario), para conocer el nombre de la interfaz creada, utilizaremos:

```
# iwconfig
```

Si, por ejemplo, la interfaz creada es **wlan0**, dado que tenemos netcfg disponible en la imagen, para configurar la red utilizaremos wifi-menu:

```
# wifi-menu wlan0
```

3. Preparar el disco duro

En GNU/Linux, los discos duros se representan con un archivo. El primer disco duro IDE (master primario) es `/dev/hda`, el esclavo primario es `/dev/hdb` y así sucesivamente. Los discos SCSI ó SATA siguen la misma convención, sólo que en forma de `/dev/sd`**`X`**.

La partición de disco es donde el sistema de archivos será creado y donde instalaremos Arch. Como mínimo se recomienda dos particiones una para la raíz del sistema de archivos (/) y otra para el área de intercambio (swap).

Para realizar la partición disponemos de varias herramientas: `cfdisk` (particiones MBR), `cgdisk` (particiones GPT) y `parted`.

CFDISK, CGDISK.- Son editores de particiones. Su interfaz está en línea de comandos y se maneja con las flechas del teclado y con la tecla Enter.

Yo he creado tres: una para la raíz (/), otra de intercambio (swap) y otra para los datos (/home).

Suponiendo que nuestro disco sea el `/dev/sda`:

```
# cfdisk /dev/sda


(/)    New-> Primary->      25GB       ->Beginning->Type->83
(swap) New-> Primary->      1GB        ->Beginning->Type->82
(/home)New-> Primary-> Resto del disco ->Beginning->Type->83
```

Nos posicionamos sobre sda1 (/) y marcamos Bootable para que se pueda iniciar desde ella.

Elegimos Write, nos pedirá confirmación, escribimos “yes” y presionamos ENTER

Formateamos las particiones, activamos swap y definimos los puntos de montaje de

nuestro sistema:

```
# mkfs.ext4/dev/sda1
# mkfs.ext4 /dev/sda3
# mkswap /dev/sda2
# swapon /dev/sda2
# mount /dev/sda1 /mnt
# mkdir /mnt/home && mount /dev/sda3 /mnt/home
```

4. Instalar del sistema base

```
# pacstrap /mnt base base-devel
```

Si disponemos de conexión cableada

```
# pacstrap /mnt ifplugd
```

Si nuestra conexión es wifi:

```
# pacstrap /mnt wireless_tools wpa_supplicant wpa_actiond dialog
```

Vamos a instalar también una utilidad para generar ***mirrors*** actualizados para el gestor de paquetes:

```
# pacstrap /mnt reflector
```

Generamos el archivo de edición de puntos de montaje, `/etc/fstab`

```
# genfstab -U /mnt >> /mnt/etc/fstab
```

5. Configurar el sistema base

Ya tenemos instalado nuestro sistema base. Ahora vamos a entrar en él

```
# arch-chroot / mnt
```

Definimos el nombre de nuestra máquina (`hostname`), en mi caso, **archninfa**:

```
# echo archninfa > /etc/ hostname
```

Para configurar el idioma, primero debemos editar `/etc/locale.gen`:

```
# nano /etc/locale.gen
```

Y descomentamos la línea correspondiente a nuestro idioma, quitando la #. Para español de España:

```
es_ES.UTF-8 UTF-8
```

Después de seleccionar el idioma, ejecutamos:

```
# locale-gen
# echo LANG="es_ES.UTF8" > /etc/locale.conf
# echo LC_TIME=es_ES.UTF8 >> /etc/locale.conf
```

Mapa del teclado:

```
# echo KEYMAP=es > /etc/vconsole.conf
```

Para la Zona horaria, podemos ver las disponibles con:

```
# ls /usr/share/zoneinfo
```

Si queremos ver las subzonas, por ejemplo, de Europa:

```
# ls /usr/share/zoneinfo/Europe/
```

Para seleccionar una, por ejemplo España:

```
# ln -s /usr/share/zoneinfo/Europe/Madrid /etc/localtime
```

Arch utiliza el reloj UTC por defecto. Si nuestro sistema va a convivir con otros, la configuración debe ser la misma en todos. Para establecer Arch como localtime (si Arch va a ser nuestro sistema principal, se aconseja configurar como UTC el resto de sistemas):

```
# hwclock --systohc --utc
```

Establecemos la contraseña de root (el usuario administrador del sistema)

```
# passwd
```

En un sistema multiusuario, no es aconsejable utilizar la cuenta **root** para todo. Su utilización debe quedar reservada a tareas administrativas. Para crear nuestro usuario, disponemos de dos métodos, `adduser` (interactivo) y `useradd`. Creemos un usuario, por ejemplo, **ninfa**, con grupo principal **users**:

```
# useradd -m -g users -s /bin/bash  ninfa
# passwd  ninfa
```

Ahora configuramos la lista de servidores para el gestor de paquetes:

```
# cp /etc/pacman.d/mirrorlist /etc/pacman.d/mirrorlist.backup

#  echo 'reflector -l 15 --sort age --save  /etc/pacman.d/mirrorlist'>
/etc/cron.daily/reflector

# chmod +x /etc/cron.daily/reflector

# /etc/cron.daily/reflector
```

6. Configurar la red

Disponemos de varios perfiles de ejemplo para configurar nuestra red en `/etc/network.d/examples/`:

- **ethernet-dhcp**: para redes cableadas con routers que dispongan de soporte DHCP.
- **ethernet-static**: para redes cableadas con IP estática.
- **wireless-wep**: para wifi con seguridad WEP.
- **wireless-wpa**: para wifi con seguridad WPA.

En mi caso, la red es wifi, la interfaz es **wlan0**, el nombre de la red es **NuestraRed**, con

seguridad **WPA**, clave **NuestraClave** y por **DHCP**. El perfil se llamará **mired**

```
# cp /etc/network.d/examples/wireless-wpa /etc/network.d/mired
```

Editamos el archivo y establecemos nuestros datos, quedando de forma similar a:

```
CONNECTION='wireless'
DESCRIPTION='A simple WPA encrypted wireless connection'
INTERFACE='wlan0'
SECURITY='wpa'
ESSID='NuestraRed'
## Uncomment if the supplied ESSID is hexadecimal
#ESSID_TYPE='hex'
KEY='NuestraClave'
IP='dhcp'
# Uncomment this if your ssid is hidden
#HIDDEN=yes
```

Para activar la conexión automática:

```
# systemctl enable net-auto-wireless.service
```

Si nuestra red es cableada:

```
# systemctl enable net-auto-wired.service
```

7. Instalar grub (gestor de arranque)

```
# pacman -S grub-bios os-prober
# grub-install --target=i386-pc --recheck /dev/sda
```

Para evitar que aparezca en el inicio un error por falta del archivo de idioma español (aún no incluido):

```
# mkdir -p /boot/grub/locale
# cp /usr/share/locale/en\@quot/LC_MESSAGES/grub.mo /boot/grub/locale/es.mo
# export LANG=es_ES.UTF-8
# mkinitcpio -p linux
```

Generaremos el menú de arranque:

```
# grub-mkconfig -o /boot/grub/grub.cfg
```

8. Reiniciar el equipo

Antes de reiniciar, activaremos el daemon cron:

```
# systemctl enable cronie.service
```

Salimos del entorno chroot, desmontamos y reiniciamos:

```
# exit
# umount /mnt/home && umount /mnt
# reboot
```

Conclusión

Cómo en todo sistema, los pasos a seguir son:

1. Configuración del teclado en el instalador (si no lo hace por defecto)
2. Configuración de la red (en otros no sería necesario, pero para Arch es imprescindible)
3. Selección y preparación del disco duro.
4. Instalación base (en algunos sistemas pueden incluir hasta los escritorios, aquí es lo mínimo para poder empezar a funcionar)
5. Configuración de sistema (idioma, zona horaria, establecimiento de la clave de root, creación del primer usuario...)
6. Instalación de un gestor de arranque.
7. Reinicio.

A partir de aquí, ya tenemos un sistema funcional. En el próximo artículo configuraremos sudo, el arranque gráfico e instalaremos un escritorio.

Referencias

https://wiki.archlinux.org/index.php/Beginners'_Guide
https://wiki.archlinux.org/index.php/Netcfg
https://wiki.archlinux.org/index.php/Systemd
https://wiki.archlinux.org/index.php/GRUB2
https://wiki.archlinux.org/index.php/Partitioning

Guifi.net: la red abierta libre y neutral

REDES

Las telecomunicaciones en nuestro país están controladas por grandes empresas monopolistas, los usuarios (sobretodo de las zonas rurales) hartos del mal servicio decidieron desplegar ellos mismos su propia red de telecomunicaciones.

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

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

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

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

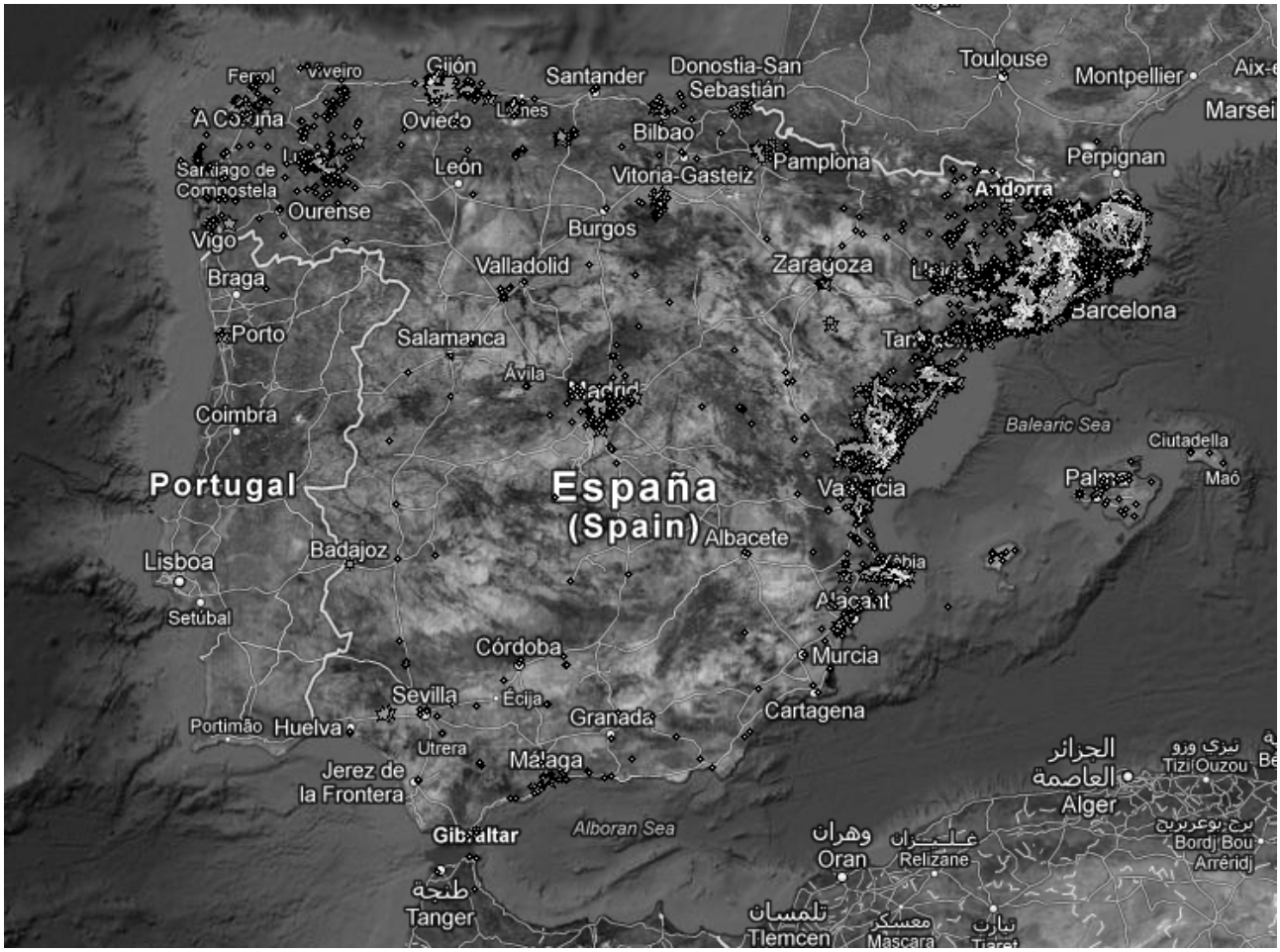
Casi parece imposible hasta donde está llegando este proyecto, pero lo que podemos ver hasta ahora es sólo el comienzo de lo que puede llegar a ser el internet del futuro. Todo empezó en 2004, cuando un hombre que vivía en el campo y trabajaba en la ciudad quería hacer una cosa tan simple como conectarse a internet desde su casa, pero al vivir en un entorno rural, dicha empresa no era tan sencilla.

Contactó con el ISP de la zona y éste se negó a ofrecerle una solución viable, básicamente, traerle una conexión a internet en su casa no era rentable. Como su vecino/hermano sí tenía conexión decidieron hacer un enlace wifi con un material carísimo que había traído Ramon Roca de uno de sus viajes comerciales a Estados Unidos y compartir la conexión a internet entre hermanos.

Esta problemática era generalizada en toda la comarca y la voz empezó a correr como el aire!

Guifi: 8 años después

Actualmente esta red ha ido creciendo y dispone de más de 18.700 nodos operativos y un número incalculable de usuarios, siendo la red wifi ciudadana mas grande del mundo.



Esto le ha otorgado al proyecto de guifi.net numerables reconocimientos como el premio de telecomunicaciones de la Generalitat de Cataluña en el 2007, en 2008 el ICT2008 lo nombró como miembro de la **European Network of Living Labs** y los dos últimos años está trabajando conjuntamente con la Unión Europea con los proyectos **CONFINE** y **BuB** (donde entre otras cosas, se estudia como tendría que ser la conectividad a "internet" en Europa de cara al futuro). Continuamente el proyecto está siendo nombrado en medios de radio y teledifusión y prensa escrita, además de los pequeños reconocimientos territoriales donde se expande.

> *BuB: Bottom Up Broadband, no esperes a que la fibra llegue a tu casa, organízate con tus vecinos, tírala y luego busca donde conectarla.*

En los últimos años los usuarios de esta red decidieron ir a mas y probar otro tipo de tecnologías, siendo ahora una de las más novedosas, la de la fibra óptica, que permite enlaces entre puntos mucho más estables y a mucha mas alta velocidad permitiendo así interconectar zonas alejadas geográficamente. Pero dicha mejora implica algunas dificultades de gestión y también de costes.

El afán de esta tecnología empezó hace ya 4 años (verano del 2008) en Gurb, la misma población a la que nos referíamos al principio de este artículo. Entre varios vecinos alejados entre ellos, decidieron comprar 2 quilómetros de fibra óptica y conectar sus Masías. La problemática pero estaba por dónde pasaban el cable; mirando el paisaje lo tuvieron claro: durante la dictadura franquista alguien plantó en sus campos unos postes que ofrecían servicio telefónico (y no mucho mas), además de que eran un estorbo al momento de labrar los campos, así que estudiaron legalmente la posibilidad de tender SU fibra óptica por dichos postes que físicamente estaban en SUS terrenos.

La ley estaba a su favor, así que lo único que tenían que hacer era hacer una notificación conforme ellos (como operador de telecomunicaciones registrado en la CMT en el 2009) iban a pasar un cable por sus postes (remarcar, unos postes ahora propiedad de una empresa privada, instalados hace muchos años por una entidad pública y pagados con el dinero de todos). Por supuesto esto no gustó nada a la empresa propietaria e hizo lo posible para poder denunciarlos y pararles los pies, pero sin éxito.

> *Guifi.net está registrado desde 2009 como operador de Telecomunicaciones en la Comisión del Mercado de Telecomunicaciones (CMT)*

Una vez conectados, tenían una intranet con una velocidad impactante (1/1Gbps), pero seguían saliendo por conexiones ADSL de muy baja calidad, así que buscaron la forma para conectar un tramo de esta fibra a la fibra óptica pública que pasaba por la carretera del lado y que justo habían acabado de reformar; inicialmente todo eran pegas e imposibles, hasta que amenazaron en cortar los 3 carriles de la carretera los fines de semana, así los que iban a esquiar, podrían pasar, pero tendrían que pasar por el arcén, representación de la frustración que sentían los habitantes de aquella tranquila zona rural con el tema de las telecomunicaciones.

A los pocos días (finales del otoño del 2009), pudieron conectar su fibra a la carretera y así llegar a Barcelona donde están comprando un Gigabit síncrono a un proveedor mayorista y lo están repartiendo entre los usuarios gracias a las asociaciones y empresas registradas como operadores de telecomunicaciones (microoperadores).

Las historias de la fibra óptica siguen, pero lo dejaremos más adelante para otro artículo más técnico :)

La red y las comunidades wireless

Tras estas historias de instalaciones de fibra óptica, que de momento son una minoría, está también otra parte técnicamente impactante y bonita del proyecto, la de la comunidad wireless, toda esta gente que de forma altruista, comercial o cooperativista, poco a poco han ido tejiendo una red mallada de más de 17.800 puntos repartidos por toda la Península Ibérica.

Un usuario si quiere conectarse a la red de guifi, lo que tiene que hacer es marcar su punto en la página web, escoger dónde se va a conectar, comprar el hardware necesario para conectarse, descargarse el fichero de configuración y conectarse a la red. A partir de ahí acceder a todos los servicios que te ofrece esta red. Previamente en la zona tiene que haber una mínima infraestructura.

Al momento de dar de alta tu nodo en la web pero, al igual que cuando instalas un software, debes aceptar unos términos de uso, la XOLN (el procomún de la Red, Abierta, Libre y Neutral), en los cuales se dictan las normas básicas de funcionamiento y uso de la red en las cuales se basan en:

1. Es abierta porque se ofrece de forma universal a la participación de todos sin ningún tipo de exclusión o discriminación y porque se informa en todo momento acerca de cómo funciona la red y sus componentes, lo que permite que cualquiera pueda mejorarla.

2. Es libre porque todos pueden hacer lo que quieran y disfrutar de las libertades tal y como se prevén en la referencia de los principios generales (apartado I.), todo esto independientemente de su nivel de participación en la red y sin imponer términos y condiciones que contradigan este acuerdo de forma unilateral.

3. Es neutral porque la red es independiente de los contenidos, no los condiciona y, así, pueden circular libremente; los usuarios pueden acceder y producir contenidos independientemente de sus posibilidades financieras o condiciones sociales. Cuando se incorporan contenidos a la red en guifi.net se hace con el fin de estimular su aparición, gestionar mejor la red o simplemente como ejercicio de incorporar contenidos, pero en ningún caso con el objetivo de sustituir o bloquear otros contenidos.

Si tras esto aún no tienes suficiente, puedes contribuir en el proyecto ayudando a otros a conectarse a la red, a mejorar las herramientas de los administradores y los usuarios (se necesitan developers!), difundiéndolo, etc.

Ponte en contacto con la comunidad![1]

[1] **Lista de correo:** https://lists.guifi.net
Foros: http://guifi.net/forum
IRC: irc.guifi.net #guifi

La importancia del shell, vi y regex

SHELL SCRIPTING

Cuando se tiene una gran familiaridad con sistemas basados en *nix, la tripleta Shell+Vi+Regex proporciona un IDE sin pretensiones. Al principio puede ser un poco confuso para los desarrolladores en lenguajes de alto nivel de abstracción, pero el dominio de estas herramientas no toma mucho tiempo y permite aprovechar al máximo los comandos del sistema operativo.

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

Desarrolladora de software, Former Head of Engineering en GiftHit (Social Gifting App), **Fundadora de Vanilla** (Billing and Revenue Management APIs). Miembro de IEEE Signal Processing Society, New York Section. "Wantrepreneur".

Webs:
About me: http://about.me/elizabethr
Vanilla: http://www.vanillaapp.co

Redes sociales:
Twitter: @eramirem

A pesar de la gran popularidad de lenguajes modernos, como Ruby y Python, del alto nivel de abstracción de la máquina y del sistema operativo que dichos lenguajes permiten y de su lenguaje natural que hace más fácil su uso, los desarrolladores de software no deben desconocer las ventajas de estar relativamente más cerca de la máquina.

Existen herramientas que se integran de manera transparente en sistemas operativos basados en *nix, que si bien no están orientadas al desarrollo de aplicaciones con interfaces de usuario sofisticadas, permiten realizar desde la ejecución de tareas básicas, a niveles avanzados de automatización y operación. Dado esto, el *shell scripting* no ha pasado de moda.

El shell

Primero que todo, para aquellos lectores que lo desconocen: ¿qué es el shell? El shell es

una herramienta de comunicación primaria con el sistema operativo a través de la línea de comandos. Al iniciar una sesión en el sistema operativo, el shell es cargado en la terminal de dicha sesión.

Existen varios tipos de shell: The Bourne Shell, el original **sh** escrito por Stephen Bourne; el C Shell, conocido como **csh**, cuya sintaxis se asimila mas al lenguaje C, haciéndolo mas uniforme con el sistema operativo; el Korn Shell, o **ksh**, el cual incluye funcionalidades de I/O mas avanzadas.

Un script en shell es un archivo que contiene un conjunto de comandos para ser ejecutados por el shell de manera secuencial. El inmenso poder de shell radica en tener todos los comandos del sistema operativo disponibles de manera nativa en tu programa. La mayoría de los comandos del sistema operativos están escritos en C, así que hablando en términos de desempeño, el shell es inigualable.

Para ilustrar las ventajas del shell en el desempeño de ciertas actividades de procesamiento de datos, pensemos en nuestro top-5 de comandos favoritos en *nix. Mi top-5 personalmente es: *grep*, *find*, *host* / *dig*, *ps*, *touch*. Ahora hagamos el ejercicio de encontrar el equivalente de estos comandos en algún lenguaje moderno.

Ejemplo: Buscar en el log del *daemon dhcpd* la *MAC address* del host que realiza una solicitud de asignación de dirección IP dinámica. Comando a utilizar: *grep*.*

1. En shell

   Sintaxis.

   ```
   grep [-abcdDEFGHhIiJLlmnOopqRSsUVvwxZ] [-A num] [-B num] [-C[num]] [-e
   pattern] [-f file] [--binary-files=value] [--color[=when]] [--
   colour[=when]] [--context[=num]] [--label] [--line-buffered] [--null]
   [pattern] [file …]
   ```

   Ejemplo.

   ```
   output=`grep -io "DHCPREQUEST.*\([0-9a-f]\{2\}:\)\{5\}[0-9a-f]\{1\}"
   ~/Documents/dhcpd.log`
   ```

2. En ruby

   Sintaxis.

   ```
   enum.grep(pattern) {| obj | block }
   ```

   Ejemplo.

   ```
   open(ENV['HOME']+'/Documents/dhcpd.log') do |f|
     f.grep(/DHCPREQUEST.*([0-9a-f]{2}:){5}[0-9a-f]{2}/) do |line|
       puts line
     end
   end
   ```

3. En python

   Sintaxis.

* grep como verbo en la vida diaria: grep for my keys, grep for my socks.

```
re.findall(pattern, string, flags=0)
```

Ejemplo.

```
import os.path
import re

f = open(os.path.join(os.path.expanduser('~'),'Documents/dhcpd.log'),
'r').read()
pattern =  r'(DHCPREQUEST.*)([0-9a-f]{2}:){5}[0-9a-f]{2}'
print re.findall(pattern, f, re.IGNORECASE)
```

En este punto, espero que los lectores hayan entendido el argumento que se plantea. Las implementaciones de los comandos del sistema operativo en diferentes lenguajes modernos, son versiones muy limitadas y carecen de los modificadores y desempeño que posee el shell. Normalmente cada lenguaje permite hacer llamadas a los comandos del shell, mediante algún método (*backticks* en Ruby, `os.system` en Python), pero con algunas limitaciones y menor desempeño.

Pipes, una de las mejores contribuciones al shell

El concepto de *pipelines* en el sistema operativo, fue creado por Douglas Mcllroy. Básicamente, los pipelines representan la combinación de múltiples comandos en una sola línea, en donde la salida del comando más a la izquierda es la entrada para el comando más a la derecha. La conexión entre los procesos ejecutados por cada comando es realizada a través de un *pipe*. La potencia de los pipes radica en que todos los procesos en el pipeline se ejecutan en paralelo, aunque su sintaxis sea serial.

Como dato curioso, los pipes fueron implementados por el genio Ken Thomson, en una sola noche.

> *He put pipes into UNIX, he put this notation into shell, all in one night. Mcllroy refiriéndose a Thomson.*

Mediante la combinación de comandos y pipes en el shell, se pueden ejecutar tareas complejas, de manera muy simple. Por ejemplo, determinar el número de conexiones al puerto 80, es tan simple como la siguiente línea:

```
netstat -an | grep :80 | wc -l
```

De igual manera que otros comandos, las implementaciones de pipes en lenguajes modernos, consisten en módulos que a su vez usan el shell, tal como la implementación de Python, pipes; restándoles así desempeño.

El impopular vi

Los usuarios de *Sublime* llaman "dinosaurios" a los usuarios de *vi*. Los usuarios de *Emacs* dicen de vi que es el editor modal del purgatorio.

Quizás el hecho que hace más impopular a vi, es que sea un editor modal, es decir, que funcione en dos modos de operación: el modo *command* y el modo *insert*. El cambio entre modos de operación es bastante molesto para los desarrolladores en IDEs modernos.

Desde luego, esto es un inconveniente menor, después de que se logran asimilar en el subconsciente los modos de operación. En algunos casos, después de usar por mucho tiempo vi, escribir un documento en un procesador de texto convencional puede ser confuso, al tratar de teclear doble d para borrar una linea.

La mayor ventaja de vi, específicamente para desarrollo en shell scripting, es que permite acceder al shell sin abandonar el editor, lo cual proporciona una integración uniforme entre sistema operativo, intérprete de comandos y editor de texto.

El editor vi se integra fácilmente con los filename wildcards del sistema operativo, lo que otros lenguajes modernos no hacen de una manera tan transparente. Por ejemplo, guardar una parte de un archivo bajo un nombre diferente al archivo en uso, es sencillo usando la notación `:n,mw filename`.

```
:1,10w ~mcastd.sh
```

El modo command permite ejecutar un gran número de funcionalidades que se integran con el shell, entre los más útiles podemos encontrar:

1. Concatenar una parte del archivo en uso a otro archivo

   Notación `:n,mw >> filename`

   ```
   :1,10w >> keep_alive.sh
   ```

2. Borrar un rango de líneas que contengan una expresión regular. Excelente para borrar líneas escritas para hacer *debug*.

   Notación `:g/regex/d`

   ```
   :g/log\sINFO/d
   ```

3. Insertar en el archivo en uso la salida de un comando del shell.

   Notación `:r! command command`

   ```
   :r! command ls -t
   ```

La mayoría de los editores modernos, permiten algunas funcionalidades similares a las que se encuentran en el modo command de vi (y sin cambiar de modo), para búsqueda y reemplazo, navegación a través de archivos, etc; mediante el uso de *shortcuts* en el teclado. Sin embargo, las limitaciones frente a vi son evidentes y el esfuerzo para aprender la notación en el modo command de vi o aprender los shortcuts de otros editores, es prácticamente el mismo.

Las potentes expresiones regulares

Una vez más, para aquellos lectores que desconozcan el tema, una expresión regular (o *regex*) es un string que contiene una combinación de caracteres normales y caracteres especiales. Los caracteres, o *metacaracteres*, hacen match con ideas, como cantidad, tipo, ubicación; entre otros. Los caracteres normales hacen match con ellos mismos. Todo esto para encontrar patrones que permitan la manipulación de texto y datos.

En las dos secciones anteriores, se observa un tema recurrente y son las expresiones regulares, para detectar o buscar patrones específicos durante el procesamiento de texto o la ejecución de un comando que recibe como argumento una expresión regular. Éstas potencian el uso de muchos de los comandos del shell y permiten navegar y sacar partido de las funcionalidades de vi. Muchos sistemas de tasación de eventos, que requieren procesamiento de eventos en tiempo real, utilizan expresiones regulares para determinar el origen y el destino de un *EDR* (Event Detail Record), para aplicar una tarifa y determinar el costo del evento, en combinación con un algoritmo de *binary search*, para encontrar de manera eficiente el patrón origen-destino que coincida con el evento recibido.

La mejor expresión regular

`[ -~]` . Esta expresión hace match con cualquier carácter ASCII desde espacio hasta circunflejo, es decir, todos los caracteres imprimibles. De esta manera, es posible realizar la validación de un argumento en muy pocos caracteres.

Conclusión

En algún momento de su vida, cualquier desarrollador podrá sentirse agradecido de saber shell, vi y regex. Si bien éstos no forman un IDE para lenguajes modernos, dichas herramientas son muy útiles para resolver problemas reales, que requieran el rescate de la potencia del sistema operativo. El shell, vi y regex no son los más aptos para construir un sitio web o una aplicación móvil. Pero para la creación de *daemons*, el análisis de procesos y datos y hacer *bootstrap* de imágenes en sistemas operativos embebidos, son una excelente opción.

Actualmente existen algunas iniciativas de emulación del shell en otros lenguajes. El proyecto nsh, es un proyecto experimental que pretende emular el shell en node.js. Para más información, visitar (o hacer fork!) https://github.com/AvianFlu/nsh

Los impresionantes archivos .PO – l10n de GNOME

SOFTWARE LIBRE - GNOME

Si eres de las personas a las que les gusta traducir o quizás traducir es tu hobby forever alone (como el mio) puedes hacer que esto sea un beneficio a los demás al participar en un proyecto de software libre contribuyendo con la localización y traducción del mismo. Encuentra aquí cómo poder hacerlo en Malditas Mentiras y en el equipo de traducción al español de GNOME. Veremos también lo impresionante y genial del rol que cumplen los archivos .PO en el proceso de traducción.

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

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

Webs:
Blog: www.milale.net

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

Si sabes otro idioma (aparte de tu lengua materna), si te gustan las traducciones (ya sea por hobby o profesión) y lo más importante si deseas colaborar en un proyecto de software libre real, hay muchas maneras de que explotes esto en un beneficio para los demás; existen muchas comunidades, proyectos y más en los cuales contribuir es un verdadero placer. En esta ocasión les diré como poder empezar a traducir para GNOME y la gran importancia que los archivos .PO tienen en este proceso.

Para empezar debes suscribirte a la página de Malditas Mentiras[0] y luego enviar un correo electrónico a la lista (en este caso es la lista al español): gnome-es-list@gnome.org donde debes presentarte y decir que deseas colaborar con los módulos.

Daniel Mustieles[1], el coordinador del equipo, te dará la bienvenida e indicaciones claras que necesitas seguir para formar parte del grupo y que te asigne el módulo del que te harás cargo en el futuro. Por ejemplo, el módulo que tengo asignado ahora es el de Funciones de gnumeric [2], aquí en la página se encuentra en detalle los porcentajes de avance y el histórico de acciones anteriores. Uno debe ingresar con su cuenta, al descargar el archivo .po de la página (se guarda directamente) y trabajamos con un editor de archivo de traducción, generalmente es Gtranslator[3].

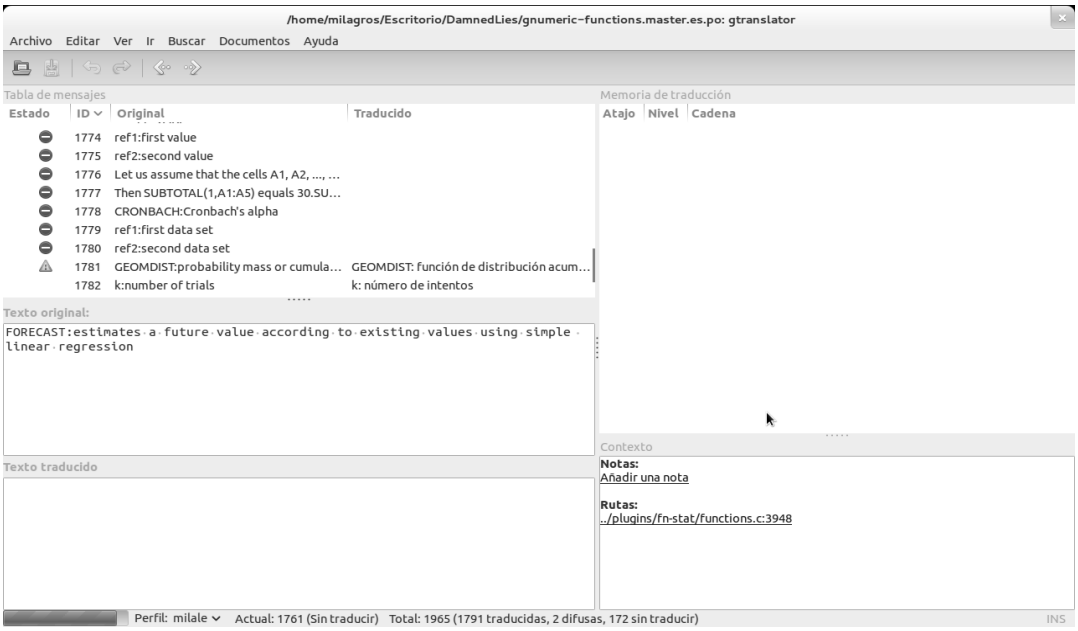
Usando Gtranslator

Gtranslator se encarga de los archivos PO gettext. Cuenta con muchas funciones como buscar, reemplazar, memoria de traducciones, tabla de mensajes, una fácil navegación, etc.

Al iniciar el programa debemos hacer las configuraciones[4] respectivas. Luego empezamos a traducir. Éste, cuenta con dos divisiones principales: en la primera se encuentra el texto original (en inglés en este caso) y debajo, la traducción realizada -o por realizar-. Los iconos de Gtranslator significan lo siguiente:

- Prohibido: la cadena está sin traducir.
- Advertencia: la cadena es difusa (esto quiere decir que la cadena estaba traducida, pero que ha cambiado algo en el código, por lo que es posible que la traducción haya que cambiarla.

Debajo, podemos ver también los detalles del traductor y del módulo, el perfil es: milale (mi nickname) y los demás datos del archivo .po con el que estás trabajando, la cadena que este seleccionada (el número y estado) y la información de las demás cadenas: el número total, las traducidas, las difusas y las que están sin traducir.



Luego de terminada la traducción, al regresar a la página de Malditas Mentiras y del módulo donde estás trabajando, en la parte de Acción nueva, primero se reserva el

módulo, así los demás sabrán que ya alguien está trabajando en el módulo. Sino, podría haber un conflicto al hacer un doble esfuerzo; después de esto ya subimos la traducción nueva, adjuntamos el archivo y al enviarlo en la lista de correos se recibe un e-mail indicando que un nuevo *commit* se realizó. Podemos ver aquí un ejemplo de cómo en la sección de Acciones aparecen las últimas, la reserva y la traducción subida al repositorio con sus *commits* respectivos.

Los escritores hacen la literatura nacional y los traductores hacen la literatura universal.
Jose Saramago.

Archivos .po

Los archivos .po (GetText Portable Objects), son archivos de texto plano en los que se almacenan cadenas de caracteres, cada una contiene el mensaje original y luego la traducción. GetText es un sistema de localización (l10n) e internacionalización (i18n) usado comúnmente para escribir programas multilingües. Se trata de un sistema unificado (texto original y traducción) por lo que los archivos resultantes suelen ser iguales; contienen el código fuente y luego se obtienen los archivos binarios compilados.

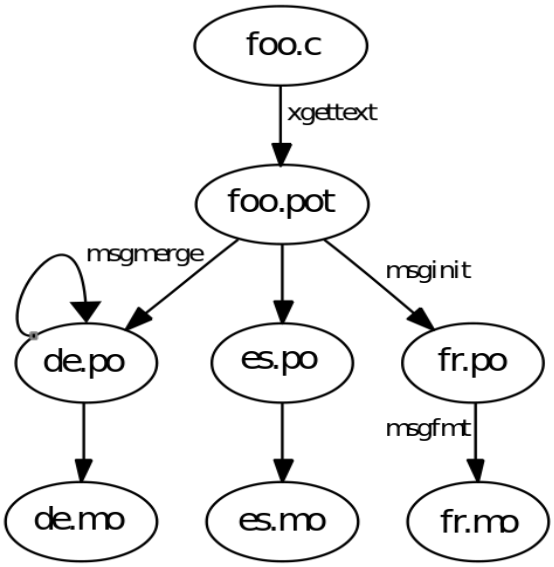
Los archivos PO, POT y MO

Para mantener las actualizaciones al día, debemos traducir los archivos .po, pero ahora veremos en detalle que son los archivos .pot y .mo también importantes.[5]

- POT (Portable Object Template): La plantilla de objeto portable es un archivo que obtienes cuando extraes textos de la aplicación, es el archivo que uno envía a los traductores.
- PO (Portable Object): Es el archivo que recibes de vuelta de los traductores, es un archivo de texto que incluye los textos originales y las traducciones.
- MO (Machine Object): Incluye el mismo exacto contenido que el archivo PO, se diferencian en el formato, solo que el PO es fácil para la lectura humana en cambio el MO está compilado y es fácil para que la computadora lo entienda; el servidor web usará el archivo MO para mostrar las traducciones.

Operación del gettext

La operación realizada es la siguiente [6]:



Programación:

El código fuente se modifica primero a usar llamadas gettext de GNU. Esto es, para la mayoría de lenguajes de programación, realizados por cadenas envueltas que el usuario verá en la función gettext. Para ahorrar tiempo escribiendo y reducir el desorden de código, esta función es usualmente asignada a _, el código C:

```
printf(gettext(“My name is %s. \n”), my_name);
```

será:

```
printf(_(“My name is %s. \n”), my_name);
```

Los comentarios (inician con ///) colocados directamente antes de las cadenas y están marcadas como sugerencias para los traductores por programas de ayuda. 'gettext' utiliza las cadenas dadas como claves para buscar traducciones alternativas y devolverá la cadena original cuando no haya traducción disponible.

'xgettext' se ejecuta en las fuentes para producir un archivo .pot o plantilla, que contiene una lista de todas las cadenas traducibles extraídas de las fuentes. Por ejemplo, un archivo de entrada con un comentario puede lucir así:

```
/// TRADUCTORES: Por favor dejar %s como está, porque es necesario para el programa.
/// Gracias por contribuir en este proyecto.
printf(_("My name is %s.\n"), my_name);
```

xgettext se ejecuta usando este comando:

```
xgettext –add-comments=/
```

El archivo .pot resultante aparecerá así, con el comentario:

```
#. TRADUCTORES: Por favor dejar %s como está, porque es necesario para el programa.
#. Gracias por contribuir en este proyecto.
#: src/name.c:36
msgid "My name is %s.\n"
msgstr ""
```

Traducción:

El traductor deriva un archivo .po de la plantilla usando el programa de msginit. Luego rellena las traducciones, msginit inicializa las traducciones, por ejemplo para una traducción al español el comando a ejecutar será (El código del lenguaje a utilizar se encuentra en el listado [7]):

```
msginit --locale=es –input=name.pot
```

Se creará el es.po. El traductor luego edita el archivo resultante, hecho a mano o con una herramienta de traducción con su modo de edición para archivos .po. Una entrada editada lucirá así:

```
#: src/name.c:36
msgid "My name is %s.\n"
msgstr “Mi nombre es %s.\n”
```

Finalmente los archivos .po están compilados en el archivo binario .mo con msgfmt.

Ejecución:

El usuario establece la variable del entorno LC_MESSAGES y el programa mostrará cadenas en el lenguaje seleccionado si hay un archivo .mo para esto.

Y para terminar...

Los archivos .po cumplen una función muy importante para el proceso de localización y traducción, es genial ver cuanto trabajo hay detrás de Malditas Mentiras, la aplicación web que gestiona la localización de GNOME. Si te gusta este mundo no dudes en animarte en ser parte del equipo y además de saber que tu contribución ayudará a que el proyecto llegue cada vez más lejos.

Referencias & Links

[0] Malditas Mentiras <http://l10n.gnome.org/about/> [Consulta: 12 de Noviembre de 2012]

[1] http://l10n.gnome.org/teams/es/

[2] http://l10n.gnome.org/vertimus/gnumeric/master/po-functions/es

[3] Gtranslator <http://projects.gnome.org/gtranslator/> [Consulta: 15 de Noviembre de 2012]

[4] http://mruiz.openminds.cl/blog/index.php/2009/02/05/configuracion-de-gtranslator-193-en-ubuntu/

[5] Archivos PO, POT y MO <http://www.icanlocalize.com/site/tutorials/how-to-translate-with-gettext-po-and-pot-files/> [Consulta: 18 de Noviembre de 2012]

[6] http://en.wikipedia.org/wiki/Gettext> [Consulta: 21 de Noviembre de 2012]

[7] http://www.gnu.org/software/gettext/manual/gettext.html#Language-Codes

Google Maps API: Primeros Pasos

GOOGLE MAPS

Google es por excelencia la empresa pionera en los sistemas de búsqueda y servicios de Internet, con una filosofía de innovación e impacto en sus servicios como AdWords, AdSense y Maps por mencionar algunos. En este artículo me enfocaré a la API versión 3 de Google Maps.

Escrito por: **Yecely Díaz** (M.I.A. & Desarrolladora Web)

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

Webs:
Blog: http://silvercorp.wordpress.com

Redes sociales:
Twitter: **@silvercorp**

Se ha convertido en una de las APIs más utilizadas debido a sus mejoras y servicios que brinda, en esta última versión también puede ser usada en dispositivos móviles y aquellos que incluyan un navegador Web con implementación de Javascript, para aquellos que no son compatibles la API proporciona imágenes de mapa en formatos PNG, JPG y GIF.

Puedes usarla en tus sitios Web comerciales y sin ánimo de lucro, en sus políticas se menciona que no puede utilizarse para identificar lugares con actividades ilegales ni para identificar información privada relacionada con personas.

Actualmente es gratuita si la utilizamos en sitios Web con un máximo de 25,000 cargas de mapas diariamente y un máximo de 2,500 cargas diarias de mapas en caso de que hayamos modificado el estilo; si rebasamos estos límites de uso deberemos adquirir una licencia Premier. Si consideras que tu sitio sobrepasa este limite deberías reconsiderar adquirir la licencia ó utilizar otros servicios como Open Street Map, Open Layers, MapQuest, entre otros.

Para este caso me enfocaré a la versión gratuita y te mostraré como utilizarla, modificar su estilo y agregar marcadores; antes que nada te menciono algunos aspectos básicos

que debes conocer.

Una API (del inglés Applicacion Programming Interface) es un conjunto de métodos que brindan una librería para ser usada por aplicaciones externas, es decir, una interfaz de comunicación. Entre las ventajas de usar una API es la integración de un servicio Web en nuestros desarrollos, la mayoría tiene métodos robustos, y nos permite olvidarnos de implementar desde cero alguna función que ya nos proporciona.

Sin embargo, debemos tomar en cuenta que por ser una aplicación de terceros dependemos de otras personas, por lo que si hay cambio en sus funciones, disponibilidad y hasta costos nos veremos en la necesidad de realizar actualizaciones o en todo caso buscar otra opción.

Los elementos básicos que debemos tener en cuenta al usar la API son:

- Agregar el código Javascript de la API
- Especificar en que contenedor será desplegado nuestro mapa
- Indicar la longitud y latitud del lugar donde deseamos posicionarnos
- Configurar el tipo de mapa, es decir si será de tipo Roadmap, Satellite, Hybrid ó Terrain.
- Y por último una función que inicialice el despliegue del mapa.

Hay más características y elementos que podemos incluir para que tu implementación sea tan completa como tu requieras.

Mi primer mapa

Es momento de conocer la API y que mejor forma que practicando y visualizando tu trabajo, imagino te preguntarás ¿que necesito?, únicamente tu editor favorito (Sublime Text, Notepad++, Coda, etc), un navegador Web (Chrome,Firefox, Safari, etc) donde visualizarás tus avances, y muchas ganas de aprender; intentaré ser lo más clara posible y puedas ver las ventajas que tiene Google Maps, el como y donde lo implementes requerirá de tu imaginación.

Sin más que decir, crea una página y guárdala, en mi caso lo llamaré index.html, la estructura básica de ella debe ser la siguiente:

```
<!DOCTYPE html>
<html>
      <head>
           <title>Mi primer mapa</title>
           <style type="text/css">
              html { height: 100% }
              body { height: 100%; }
              #contenedor_mapa { margin: auto 0 auto 0; } /*Contenedor*/
           </style>
      </head>
      <body>
            <p>Utilizando la API de Google Maps</p>
```

```
      </body>
</html>
```

Debido que la API está desarrollada en Javascript usaremos la etiqueta <script></script> para incluir el código necesario para nuestra aplicación, como es la integración de la API a nuestro proyecto por lo que agregamos la siguiente línea:

```
<script type="text/javascript" src="https://maps.google.com/maps/api/js?
sensor=set_to_true_or_false"></script>
```

Puedes definir el sensor en *true* si la aplicación será la que determine la ubicación del usuario o *false* en caso contrario; para este ejemplo lo dejaremos con un valor de false. Es momento de definir las características de nuestro mapa, como te mencioné al ser Javascript iniciaremos el código de la siguiente manera:

```
<script type="text/javascript">
  function inicializarMapa() {
      var coordenadas = new google.maps.LatLng(24.634504, -104.552784);
      var opciones = {
            zoom:      5,
            center:    coordenadas,
            mapTypeId: google.maps.MapTypeId.ROADMAP
      }
      var mapa = new
google.maps.Map(document.getElementById("contenedor_mapa"), opciones);
  }
</script>
```

La variable **coordenadas** tiene los valores de longitud y latitud correspondientes a la posición en el mapa, el objeto **opciones** indica que realizará un zoom de 5, lo centrará en esas coordenadas y el mapa será de tipo ROADMAP; si quisieras seleccionar otro tipo, por ejemplo Satellite solo deberás cambiar la instrucción a google.maps.MapTypeId.SATELLITE.

Y por último creamos el objeto mapa indicando que será inicializado y desplegado en el elemento "contenedor_mapa", así como hacer la llamada de inicializarMapa en el evento onload de tu <body>, es decir:

```
<!DOCTYPE html>
<html>
       <head>
              <title>Mi primer mapa</title>
              <style type="text/css">
                     html { height: 100% }
              body { height: 100%; }
              #contenedor_mapa { margin: auto 0 auto 0; } /*Contenedor*/
              </style>
              <script type="text/javascript"
                   src="https://maps.google.com/maps/api/js?sensor=false"></script>

       <script type="text/javascript">
              function inicializarMapa() {
                     /* Primer parámetro corresponde a latitud y el segundo a la longitud */
                     var coordenadas = new google.maps.LatLng(24.634504, -104.552784);

                 var opciones = {
                     zoom:      5, /*El zoom puede iniciar en 1 */
                      center:    coordenadas,
                      mapTypeId: google.maps.MapTypeId.ROADMAP
```

```
                    }
                    var mapa = new google.maps.Map(document.getElementById("contenedor_mapa"),
                          opciones);
              }
        </script>

        </head>

        <body onload="inicializarMapa()">
              <p>Utilizando la API de Google Maps</p>
              <div id="contenedor_mapa" style="width:100%; height:100%"></div>
        </body>
</html>
```

"Sin requerimientos o diseño la programación es el arte de agregar errores a un archivo de texto vacío" - Louis Srygley

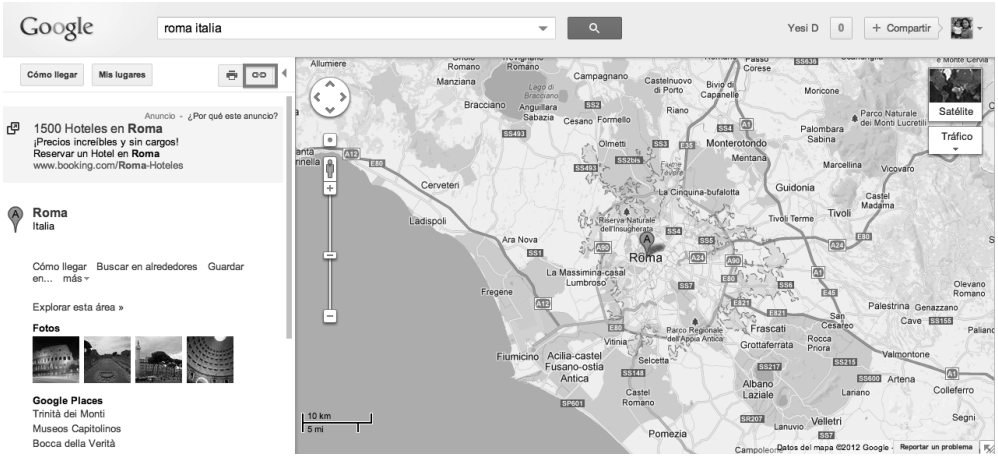
¿Cómo obtener las coordenadas?

En el ejemplo anterior estoy apuntando hacia México, pero que pasa si tu deseas cambiar la ubicación, solo sigue estos sencillos pasos para obtener la longitud y latitud.

1. Entra a Google Maps

2. En su buscador escribe el lugar que tu desees, para el ejemplo buscaré las coordenadas de Roma.

3. Una vez que tienes el resultado y se te presenta en el lado derecho de la página (ver siguiente figura) da clic en el icono de "link".



4. Copia y pega en tu editor, la dirección que te aparece (http://maps.google.com/maps?q=roma&hl=en&sll=**23.634501,-102.552784**&sspn=21.431695,43.286133&hnear=Rome,+Province+of+Rome,+Lazio,+Italy&t=m&z=10)

5. Como puedes ver tienes muchos datos en esa URL, únicamente es útil para nosotros lo que se encuentra después del campo sll que he marcado en negritas y hasta antes de

&sspn.

6. Copia esas coordenadas y sustitúyelas (si así lo deseas) por las que yo ingresé de México.

7. Actualiza y ve la nueva ubicación.

Hasta aquí ya tienes tu primer mapa y sus opciones, sigamos con la creación de marcadores.

¿Cómo agregar marcadores al mapa?

La implementación puedes hacerla tan completa como tu desees y una opción más que tienes es agregar elementos a tu mapa como son los marcadores, polilíneas, ventanas de información, círculos, etc.

Los marcadores pueden mostrar el icono que tu le asignes o en caso contrario el asignado por defecto, la instrucción es *google.maps.Marker* siendo posible configurar su posición, icono, título, animación y la propiedad de "draggable"; lo siguiente será añadir en tu código después de que creaste tu mapa (ver mapa) un marcador simple por lo que crearemos el siguiente objeto:

```
var marcador = new google.maps.Marker({
      position: coordenadas,
      title: "Mi primer marcador"
});

marcador.setMap(mapa);
```

Estamos indicando que se posicionará en las coordenadas que estableciste y como título "Mi primer marcador", la última línea se encarga de añadir el marcador a tu mapa. ¿Y cómo cambiamos el icono? tiene una propiedad denominada "icon" y debes indicar la ruta donde se encuentra tu imagen, por ejemplo si está situada en tu carpeta imágenes y se llama `miIcono.png` deberás definirlo así:

```
var marcador = new google.maps.Marker({
      position: coordenadas,
      title: "Mi primer marcador",
      icon: 'imagenes/miIcono.png'
});

marcador.setMap(mapa);
```

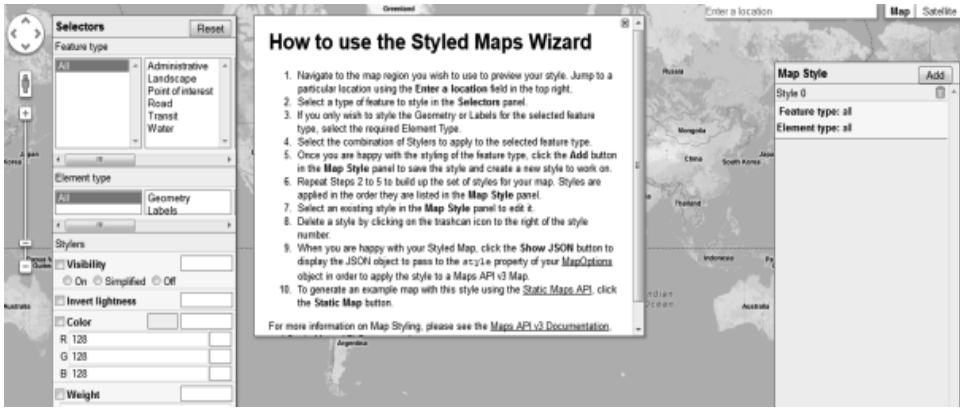
El siguiente fragmento de código tiene 2 nuevas propiedades (draggable y animation), la primera se asigna en true y al darle clic a tu icono podrás situarlo en cualquier lugar de tu mapa y la segunda opción realiza el efecto de caer desde la parte superior de tu mapa.

```
var marcador = new google.maps.Marker({
      position: coordenadas,
      title: "Mi primer marcador",
      icon: 'imagenes/miIcono.png',
```

```
        draggable: true,
        animation: google.maps.Animation.DROP
});

marcador.setMap(mapa);
```

Cambiando la apariencia de nuestro mapa

Si deseas personalizar aún más la apariencia de tu mapa Google te brinda una herramienta llamada "Styled Map Wizard" donde visualmente podemos cambiar la configuración y estilizarlo como queramos, únicamente entra a la página y verás una pantalla como la siguiente:



Su ventaja principal es que tu mapa lo podrás personalizar de una manera sencilla, a continuación te explicaré como realizarlo.

1. En la primera opción verás la propiedad de **Featured type**, es decir, todos los recursos que podrás visualizar en tu mapa como son escuelas, hospitales, etc; debes tener en cuenta que estas opciones solo funcionan para algunos países.
2. La opción siguiente denominada **Element type** tiene el subconjunto de recursos aplicables al mapa.
3. En **Stylers** puedes jugar con los valores referentes al color, gama de colores, saturación, etc.
4. En cuanto termines de asignar a cada opción sus valores y te encuentres satisfecho de tu mapa es momento de crear el archivo JSon. De lado derecho en la parte inferior verás un botón llamado **"Show Json"**, al dar clic desplegará datos similares a los siguientes:

```
[
      { "elementType": "geometry",
        "stylers": [ { "hue": "#0077ff" }, { "saturation": 13
            { "lightness": 22 }, { "gamma": 0.79 }, { "weight": 0.5 },
            { "visibility": "simplified" }
         ]
      }
]
```

5. En tu función analizarMapa() pega el código que generaste y agrega la siguiente línea:

```
var estilo = new google.maps.StyledMapType(miEstilo, {name: “Styled Map”});
```

6. Agrega a tus opciones el valor mapTypeControlOptions y el estilo.

```
MapTypeControlOptions: {
   mapTypeIds: [google.maps.MapTypeId.ROADMAP, 'estilo']
}
```

7. Posicionate después de la línea donde creas tu mapa (var mapa) y escribe

```
mapa.mapTypes.set('estilo', estilo); /*Asignando el estilo*/
mapa.setMapTypeId('estilo');
```

El código final de este artículo debe ser similar al que te presento a continuación:

```
<!DOCTYPE html>
<html>
	<head>
		<title>Mi primer mapa</title>
		<script type="text/javascript"
		     src="https://maps.google.com/maps/api/js?sensor=false"></script>
		<style type="text/css">
			html { height: 100% }
		body { height: 100%; }
		#contenedor_mapa { margin: auto 0 auto 0; }
		</style>
	</head>
	<script type="text/javascript">
		function inicializarMapa() {
			/* Primer parámetro corresponde a latitud y el segundo a la longitud */
			var coordenadas = new google.maps.LatLng(24.634504, -104.552784);

			var miEstilo = [
				{ "elementType": "geometry",
					"stylers": [
						{ "hue": "#0077ff" },
						{ "saturation": 13 },
						{ "lightness": 22 },
						{ "gamma": 0.79 },
						{ "weight": 0.5 },
						{ "visibility": "simplified" }
				   ]
				}
			];

			var estilo = new google.maps.StyledMapType(miEstilo,
				{name: 'Styled Map'});

		    var opciones = {
			   zoom:      5, /*El zoom puede iniciar en 1 */
			  center:    coordenadas,
			  MapTypeControlOptions: {
			     mapTypeIds: [google.maps.MapTypeId.ROADMAP, 'estilo']
			   }
		    }

		    var mapa = new google.maps.Map(document.getElementById("contenedor_mapa"),
				opciones);

		    /*Asignando el estilo*/
		    mapa.mapTypes.set('estilo', estilo);

		    mapa.setMapTypeId('estilo');

		    var marcador = new google.maps.Marker({
```

```
                        position: coordenadas,
                        title: "Mi primer marcador",
                        icon: 'imagenes/miIcono.png',
                        draggable: true,
                        animation: google.maps.Animation.DROP
                    });

                    marcador.setMap(mapa);

                }
                </script>
        <body onload="inicializarMapa()">
                <p>Utilizando la API de Google Maps</p>
                <div id="contenedor_mapa" style="width:100%; height:100%"></div>
        </body>
</html>
```

El código anterior así como el icono podrás encontrarlo en mi cuenta de GitHub, en un siguiente artículo veremos como crear marcadores múltiples, polígonos, círculos, etc.

THREE.JS ¿va a hacer todo eso por mi?

JAVASCRIPT

En esta ocasión haremos el intento de un pseudo-tutorial súper básico, viendo paso a paso lo lindo de THREE.JS. ¿Qué es esto? Es una biblioteca que nos permite abstraernos bastante bien de los inconvenientes a la hora de jugar con 3D en nuestros *browsers.*

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

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

Webs:
Blog: http://yetanotherlog.wordpress.com/

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

Three.js, como nos dice la wiki [1] es una biblioteca concebida con el fin de crear y visualizar gráficos 3D en nuestros navegadores. Three.js puede interactuar tranquilamente con elementos de HTML5, SVG o WebGL. El código junto con excelentes ejemplos, está disponible en *GitHub* [2]. Esto, sumado a un mínimo/moderado conocimiento de cuestiones gráficas nos permitirá crear aplicaciones (desde visualizaciones científicas hasta juegos) muy fácilmente.

Encendiendo Motores ...

En nuestro experimento vamos a querer tener unas cuantas cosas:

- ***Escena***, un lugar donde poner todos nuestros elementos (cámara y objetos «entre otros yuyos»[1]).
- ***Render***, el que genera la imagen desde el modelo que le damos.
- ***Cámara***, para hacerlo sencillo, pensemos que nos muestra sigilosamente a nuestros objetos desde una cierta posición.
- ***Luces,*** para darle una iluminación determinada a nuestra escena.
- Un ***objeto***, al cual mirar detenidamente :)

Escena, Cámara y Render

Ahora pasemos lo comentado anteriormente a código, con la ayuda de nuestra biblioteca mágica. Antes que nada tenemos que traernos un *container* a quien le vamos a "enchufar" nuestro render de la siguiente manera:

```
container = document.getElementById('container');

// creamos nuestra escena
scene = new THREE.Scene();

// creamos la camara indicando por parametros el ángulo de la vista
// entre otros y le damos su posicion en el eje z
camera = new THREE.PerspectiveCamera(45, window.innerWidth /
window.innerHeight, 1, 10000);
camera.position.z = 500;

scene.add(camera);

// creamos nuestro render WebGL y se lo agregamos a nuestro container
renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);

container.appendChild(renderer.domElement);
```

Por lo que a la hora de dibujar en la pantalla deberemos llamar a nuestro *renderer* de la

1 NdR: la expresión "entre otros yuyos" es un argentinismo utilizado para expresar que lo que antedicho, forma parte no exclusiva ni excluyente, de un todo más amplio y abarcador. El término "yuyo" se utiliza en varias regiones de América del Sur (Argentina, Chile, Paraguay, Uruguay y Bolivia), para referirse a la "hierba mala", una planta herbácea que crece espontáneamente dificultando el buen desarrollo de los cultivos.

siguiente manera:

```
renderer.render(scene, camera);
```

Objetos y Luces

Ahora que ya tenemos nuestra cámara, escena y render pasemos a colocar un objeto!

```
// creamos una simple esfera con un radio = 100  con segmentos y
// anillos = 16
var sphereMaterial = new THREE.MeshPhongMaterial({ambient: 0x000000, color:
0xffffff, specular: 0x555555, shininess: 10});
var sphere = new THREE.Mesh(new THREE.SphereGeometry(100, 16, 16),
sphereMaterial);

// Y no se olviden de agregarlo a la escena
scene.add(sphere);
```

A continuación, querremos asignarle un material a nuestro objeto. ¿Qué significa esto? Es la apariencia que deseamos que tenga nuestra esfera. Explicar el ***Modelo de reflexión de Phong*** supera el fin de este artículo pero pueden ver de que se trata en nuestra querida Wiki [3] y si intentan implementarlo por su cuenta podrán apreciar el trabajo que *THREE.JS* hizo por ustedes :)

Ahora veamos otra cuestión que trata *THREE.JS* de forma muy agradable .. la iluminación. Podemos crear distintos tipos de luces, posicionarlas y regular su intensidad muy fácilmente.

```
pointlight = new THREE.PointLight(0xFFFFFF);
ambientlight = new THREE.AmbientLight(0x101010);

// este 1 nos da la intensidad. Si se desea que sea más leve basta con
// colocar 0.5
directlight = new THREE.DirectionalLight(0xFFFFFF, 1);

// aquí definimos de donde proviene la luz
directlight.position.set(1, 1, 0).normalize();

// Y como siempre debemos agregar las luces a la escena
scene.add(pointlight);
scene.add(ambientlight);
scene.add(directlight);
```

Si deseamos que uno de nuestros objetos “emita” luz podemos hacer lo siguiente:

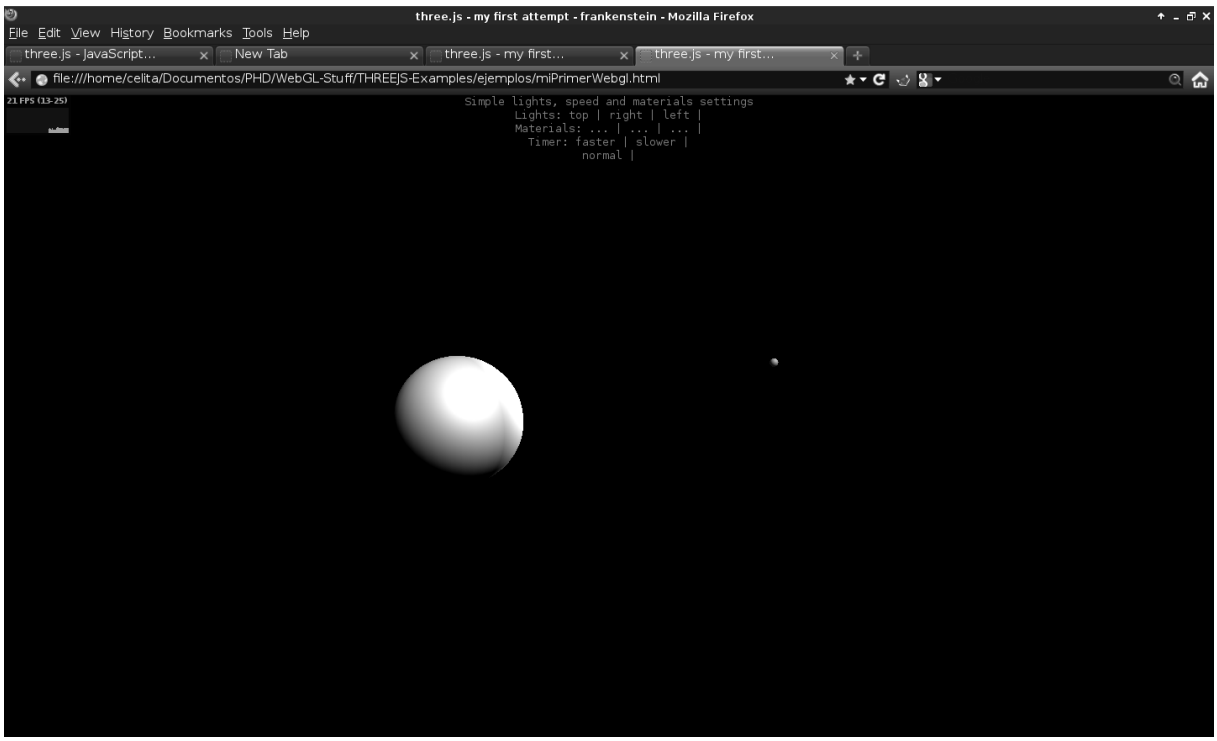
```
// Creamos nuestra esfera como lo vimos anteriormente
lightmesh = new THREE.Mesh(new THREE.SphereGeometry(10, 16, 8), new
THREE.MeshPhongMaterial( {color: 0xFFAA00}));

// A nuestra nueva esfera le asignamos la posición de luz que generamos
// renglones atrás.
lightmesh.position = pointlight.position;

// Y la agregamos a la escena
 scene.add(lightmesh);
```

Animaciones

En nuestro ejemplo queremos colocar la esfera en una posición y la segunda esfera que ilumina, gire alrededor de ella con una cierta órbita. Pueden ver un pequeño vídeo de como se vería en http://www.youtube.com/watch?v=Z3w-lLeetmw.

Para animar cualquier objeto, simplemente tenemos que modificar su posición cada vez que vayamos a dibujarlo. Por lo que a nuestra pequeña esfera le diremos que se modifique (antes de llamar a nuestra funcion render) con:

```
var timer = speed * Date.now();

lightmesh.position.x = 1000 * Math.cos(timer);
lightmesh.position.z = 1000 * Math.sin(timer);
```



La variable *speed* nos permitirá acelerar o disminuir la velocidad de nuestra pequeña esfera.

El ejemplo completo lo pueden encontrar en https://github.com/celiacintas/WebGL-Stuff/blob/master/THREEJS-Examples/ejemplos/miPrimerWebgl.html. En éste, podrán observar botones para cambiar la posición de alguna de las luces, la velocidad y cambios de la posición de la cámara con movimientos del mouse.

Vale repetir y machacar que en el repositorio de *THREE.JS* encontrarán muchísimos más ejemplos y en su página, un pequeño tutorial y su documentación.

Referencias & Links

[1] Wikipedia THREE.JS
[2] Repositorio THREE.JS
[3] Modelo de Reflexión de Phong
[4] Video del Ejemplo
[5] Codigo Fuente del Ejemplo

Arquitectos y diseñadores, los roles opcionales

GESTIÓN DE PROYECTOS

La necesidad de generar un negocio rentable a partir de la construcción de Software, ha llevado a casas de Software y áreas de sistemas de empresas de todos los tipos, a sacrificar partes y roles del proceso Software que "parecen" (a su criterio) no aportar en la materialización del producto Software. Es así como hoy vemos muchos proyectos en los cuales se omiten prácticas de Arquitectura y Diseño de Software, pero lo más preocupante es que en algunos casos no es por elección sino por desconocimiento acerca de las responsabilidades de estos roles y de los beneficios de contar con ellos dentro del proceso Software.

Escrito por: **Sorey García** (Especialista en Desarrollo de Software)

Arquitecta de Software y docente universitaria. **Fundadora de la comunidad Avanet** y miembro de **IEEE Subsección Medellín**. **Líder de investigación y desarrollo** para *movilidad* y *retail* en **LineaDataScan**.

Webs:
Blog: http://blog.soreygarcia.me

Redes sociales:
Twitter / Identi.ca: **@soreygarcia**
LinkedIn: http://co.linkedin.com/in/soreygarcia

Muchos proyectos hoy en día se afrontan sin arquitectos y/o diseñadores de Software, o bien, sin personas que a pesar de no tener asignado el rol explícito, cuenten con las habilidades que requieren quienes ejecutan sus tareas. Los roles que ya se asumen en la construcción de Software sin mayor problema son: *los clientes, los gerentes de proyectos, los analistas de requisitos y los programadores*, en algunos casos con buena suerte, se involucran además a los testers o probadores y a los

usuarios. Uno diría que es un buen avance, pero a la vez que **después de todo el tiempo invertido por parte de las organizaciones en intentar hacer ingeniería de Software, también es un gran atraso,** en mayor medida por que en muchos de los casos, su omisión se basa en desconocer cuales son las responsabilidades y beneficios de contar con quien ejerza esos roles dentro de los proyectos y no en una elección consciente.

Desde hace algún tiempo he sostenido con algunos de mis colegas arquitectos y diseñadores varias conversaciones acerca del motivo por el cual ocurre esta situación, en la que arquitectos y/o diseñadores de Software se convierten en **roles opcionales**, los segundos más que los primeros. La conclusión más simple es que si existen serios problemas en los roles más comunes y conocidos, como el del programador, los más "*desconocidos*" evidentemente tienen más problemas, sin embargo **el desconocimiento no es ni mucho menos la mejor de las excusas** y, el principal problema es el mismo: la falta de madurez de la profesión en muchos lugares del mundo.

Una de las causas del problema se encuentra en las mismas universidades, que en ocasiones ni mencionan estos roles y mucho menos forman personas con las habilidades necesarias para ejercerlos y, es que igual uno se preguntaría *¿cuántos docentes tienen el conocimiento suficiente para enseñar diseño de Software útil empresarialmente y no simplemente teórico? O bien, ¿están los estudiantes universitarios de pregrado preparados para entender el valor de estas disciplinas o sería tiempo perdido?*

Por otro lado es necesario entender que las habilidades de un arquitecto o diseñador están lejos de tener que ver la metodología que se ejecuta. Si bien, en unas u otras la documentación de este tipo de ejercicios es menor o mayor, tomarse el tiempo de *abstraer las necesidades de usuarios y negocios, identificar una o varias soluciones previas antes de dar inicio a la construcción e incluso detallar aquellas que resulten críticas o tengan un desarrollo complejo,* son algunas de esas prácticas que podrían beneficiar en mucho los proyectos, pero al ser desconocidas los son también todos sus posibles beneficios.

Uno de los errores típicos en las empresas empieza por pensar que el desarrollador que más tiempo lleva en la compañía es quien debe ejercer el rol de arquitecto. Para muchos de nosotros es claro que las habilidades para realizar este tipo de actividades no necesariamente están presentes por tener habilidades para la programación, aunque se haya programado muchos años o aunque se sea muy hábil en ello.

Algunos programadores no adquieren las habilidades requeridas como son *el liderazgo, la visión de negocio, la negociación de conflictos, las habilidades de comunicación y una muy importante y básica, la abstracción*, que si bien está implícita en las labores de un programador, cuando se trata de entender los detalles de los escenarios de negocio para encontrar el foco de un problema, identificar las partes más relevantes y pensar en la solución sin tener necesariamente que entrar en la construcción de la misma, **son habilidades que requieren de experiencia y madurez profesional.**

El rol del diseñador si bien va a un nivel de detalle mayor que el de arquitecto, es igualmente importante y, en esa medida debe ser ejercido coherentemente y por la persona indicada, puesto que este personaje se convierte en el centro de gran número de actividades. El diseñador es la persona encargada de comprender en detalle cada uno de los requisitos y de acuerdo a los lineamientos establecidos por el arquitecto, plantear de forma detallada como se resolverán e implementarán cada una de las funcionalidades

del sistema. Además, debe estar en capacidad de discrepar -si se diera la situación- las indicaciones del arquitecto y demostrar técnicamente que una solución no es correcta o en cualquier caso, hacer pruebas de concepto para validar alguna indicación arquitectónica de la cual desconoce su efecto.

En esta medida el diseñador termina convirtiéndose en muchos casos en el líder técnico y gran apoyo de los probadores, puesto que es quien tiene funcional y técnicamente el contexto del proyecto en su mente y quien lo refleja en la documentación que decida construirse. Así pues, si bien necesitamos alguien con las mismas características que un arquitecto, **es más fácil llevar paulatinamente a un buen, disciplinado y experimentado programador a ejercer este rol y cultivar su experiencia, para ser posteriormente un buen arquitecto.**

> *Los arquitectos y diseñadores de Software, como el buen vino, toman tiempo en madurar.*

De similitudes con las edificaciones

La ingeniería de Software ha nacido de la necesidad y exigencia a los ingenieros de Software de construir sistemas con procesos similares a los de una fabrica o tal cual como se construyen las edificaciones. La solicitud no es más que el llegar a ser mas exactos, producir más elementos en serie y equivocarse mucho menos de lo que venimos equivocándonos de acuerdo al popular *CHAOS Report de Standish Group*.

	1994	1996	1998	2000	2002	2004	2006	2009
Exitoso	16%	27%	26%	28%	34%	29%	35%	32%
Parcialmente Exitoso	53%	33%	46%	49%	51%	53%	46%	44%
Fallido	31%	40%	28%	23%	15%	18%	19%	24%

Cabe aclarar que en la tabla anterior, con la clasificación proyectos “exitosos”, *Standish Group* se refiere a aquellos que fueron terminados a tiempo y con el presupuesto esperado. Mejorar estas cifras es académica y empresarialmente un reto, evidentemente más que por solo mejorar las cifras, por evitar los efectos económicos que generan los atrasos y el uso desmedido de recursos.

Pues bien, actualmente es claro para la mayoría de nosotros, que hacer Software no se parece mucho a ser una fábrica o construir un edificio, debido a que a diferencia de estos escenarios, **la mayor parte de la materia prima se basa en las habilidades humanas de los involucrados, que están lejos de poder ser medidas y estandarizadas,** y que son susceptibles a cosas tan simples como el humor, el clima, el entorno o las relaciones entre los integrantes del equipo. Sin embargo, algunas comparaciones claves con estos escenarios ayudan a llevar fácilmente los temas de arquitectura y diseño a las mentes de quienes recién comienzan, o a justificarlo frente a quienes no tienen la capacidad de entenderlo. Es claro igual, que no deben perderse de vista las serias diferencias, una de

ellas bastante gráfica y sencilla de entender, es como un arquitecto de edificaciones puede determinar por ejemplo cuantos ladrillos, acero y cemento podría tomar a un albañil construir una pared y como un arquitecto de Software jamás estará en posibilidad de determinar cuantas líneas de código tomará a un programador construir un componente de Software.

Ahora bien, siguiendo con las comparaciones en las que sí ayuda establecer puntos comunes entre la construcción de edificaciones y la construcción de Software, uno de los puntos más importantes es el modelamiento o los planos que se construyen. Pocos de nosotros imaginaríamos una cuadrilla de albañiles construyendo un edificio de 50 pisos sin que antes un arquitecto, un topógrafo, un delineante de arquitectura y quien sabe cuantos roles más, se hubiesen tomado el trabajo de estudiar el terreno, hacer maquetas y planos detallados, sin embargo, es bastante común ver proyectos de Software grandes y pequeños que fueron analizados, *no* diseñados, construidos y probados por una cuadrilla de programadores junior, bien sea por costos, por desconocimiento o por simple irresponsabilidad. Sin embargo, pocos se alarman ante dichas situaciones.

No entraré a detallar los puntos acerca de si según el tamaño de la edificación esto podría hacerse sin problemas, solo les dejaré un ejercicio práctico: la próxima vez que adquieran un mueble con instrucciones de "hágalo usted mismo" intente hacerlo sin mirar un solo instante el pequeño manual de instrucciones que le adjuntan y luego trasporte dicho escenario a los dueños de los productos de Software grandes y pequeños que finalmente son sus clientes, tal cual como usted se sienta, así "deberían" sentirse ellos, aunque para su tranquilidad y mi lástima, ellos no son muy conscientes de que alguien sin un plan está tratando de construir un producto que hará una labor importante en su negocio.

Continuando con el tema del modelamiento, hay varias propuestas. No soy yo quien va a decirles si considera alguna más adecuada que otra, **sin embargo si usted considera los deseos de sus usuarios como un laberinto oscuro y complejo, mínimamente debería equiparse en la aventura con un mapa y varios expertos en exploración.**

Personalmente considero que el modelamiento de Software es una de esas actividades que sin madurar lo suficiente, fue y es subestimada hoy en día, **la idea de poder dibujar tus ideas en un lenguaje común que entiendan los diferentes participantes de un proyecto, no es en absoluto descabellada**. Sin embargo dos cosas satanizan los buenos esfuerzos realizados al respecto: la extrema complejidad de proyectos desarrollados bajo RUP o la rigidez del modelo en cascada, exigiendo que todo el diseño fuese terminado antes de empezar a construir sin estar preparados para los cambios, tema que es mejor cubierto por las metodologías ágiles, las cuales lastimosamente son tomadas por muchos irresponsablemente, como excusa para omitir muchas prácticas positivas del proceso Software, situación que para nada se parece a las verdaderas propuestas de quienes evangelizan con respecto a ellas.

Es el momento de entender que tomarse el Software cómo un deporte no le hace bien a ningún proyecto y que es mejor realizar buenos productos a simplemente sentarse a rezar por que ninguna de nuestras edificaciones (entendidas como nuestros productos de Software) se desplome alguna vez y produzca una catástrofe.

Por otro lado, haciendo alusión nuevamente al tema de las habilidades de arquitectos y diseñadores en esta comparación con las edificaciones, les dejo una reflexión de un foro

en la **XII Conferencia Iberoamericana de Ingeniería de Requisitos y Ambientes de Software** que lo ilustra perfectamente, apelando al hecho de que muchos, además quieran formarse como arquitectos con solo lectura y aulas de clase:

> *...es fácil pensar en que un Arquitecto de edificaciones puede ser bueno sin tener que haber pegado jamás un ladrillo sobre otro, pero no podemos pensar lo mismo de un Arquitecto de Software que no haya programado jamás…*
>
> ***IDEAS '09***

De modelos de negocio

Hasta este momento he abordado el tema de la carencia o ausencia de los roles de arquitecto y diseñador desde el desconocimiento, pero para finalizar me gustaría dejarles otro escenario **¿Qué tal si la decisión de no contar con estos roles es una decisión consciente?**

Si uno analiza o vive los escenarios en los que sí se ejecutan los procesos contando con arquitectos expertos y diseñadores hábiles, termina por concluir varias cosas, entre ellas que en este escenario se necesitan muchos menos programadores experimentados y en consecuencia que se puede adquirir fuera laboral más económica. La razón es sencilla, para eso se ejecutaron las tareas de arquitectura y diseño detallado.

Este escenario puede incomodar bastante a los programadores más creativos y hábiles, uno puede observar claramente la desazón de un programador que trabaja siguiendo las directrices de un arquitecto y diseñador, su libertad y creatividad claramente se ven coartadas al llevar un proceso que lo obliga a seguir con suma precisión las indicaciones de alguien más, sin embargo a nivel de empresa es un excelente modelo, pues las habilidades de los programadores no tienen que ser tantas, ya que se limitan a seguir instrucciones. Los programadores más hábiles y creativos, que son típicamente la excepción y no la regla, son enfocados a seguir creciendo como diseñadores y arquitectos y obtienen mayores remuneraciones económicas. Entre tanto, la empresa contrata programadores que en ocasiones pueden ser incluso estudiantes, ya que se van a limitar exclusivamente a programar y su carencia de habilidades en el tema será guiada y supervisada por arquitectos, diseñadores y líderes técnicos.

En el negocio opuesto, aquellas casas de Software que trabajan sin diseñadores y arquitectos asignados a los proyectos, se tendrá que tener excelentes desarrolladores que además de ser bastante independientes y disciplinados, sean hábiles para determinar la solución a cualquiera de los problemas que se presenten, además de poder analizar problemas, determinar las soluciones, comunicarlas al cliente y por supuesto documentarlas e implementarlas. En este escenario se cuenta por ejemplo con algún grupo general de especialistas o arquitectos que apoyen los proyectos al inicio o cuando

estos entran en dificultades. Parece un escenario tan válido como el anterior, pero en este caso además de que los programadores deben tener salarios superiores, deben tener múltiples habilidades para ejercer sus múltiples responsabilidades y conseguir de esos, no está nada fácil hoy por hoy.

Ahora bien, abogando siempre por la ética sobre el tema, **¿son los clientes conscientes de que el modelo de negocio que decide la casa de Software que contrata incide en la mantenibilidad de los productos que le construyen?**

Si una persona tiene demasiadas actividades de diferentes roles, con severa dificultad entrará a realizar todos los artefactos propuestos por algunas metodologías, caso contrario a si tienes una persona con un rol asignado cuya labor y foco se centra en delimitar el camino a otros, evidentemente muchos más artefactos serán generados.

Obviamente, más artefactos generados serán más cosas que mantener en el futuro, pero **¿quién no ha escuchado alguna vez a un grupo de desarrolladores sugerir reconstruir un Software que tomó años en ser estabilizado, solo porque no lo entienden?** La decisión es sobre el futuro y como finalmente se trata de un servicio, pagar o no pagar por la generación de artefactos es decisión exclusiva de los clientes.

Lo que debe ser claro es que los principios del modelamiento en ningún momento dicen que deban construirse cientos de diagramas por que sí, de hecho uno debería construir expresamente lo que necesita para que exista una comprensión suficiente para los participantes del proyecto. Mi pensamiento bastante personal es que **en el momento que sientes que estás construyendo un artefacto cualquiera, únicamente por seguir una metodología o por una indicación de la empresa, pero ese artefacto no te aporta absolutamente nada, entonces vas por el camino equivocado.** Pero elegir omitir la construcción de los mismos solo por no saber hacerlos o desconocer lo que comunican, puede simplemente estar generando una omisión que con seguridad no se pagará en el presente, sino en el futuro por parte de los clientes cuando sus proyectos tengan problemas de mantenibilidad.

Los escenarios están planteados. Son decisiones tanto de los proveedores de soluciones como de los desarrolladores involucrados en el proyecto el ejecutar o no labores de arquitectura y diseño y plantear su importancia a los clientes para que decidan si desean o no pagar por ellas, pero como les planteé en mi primer artículo en el número anterior de nuestra revista, quizá es el momento de que las demás personas sean educadas en los menesteres de construir Software y que entiendan de verdad como los afectan las decisiones técnicas con respecto a las prácticas ejercidas y no ejercidas por los ingenieros de Software, por omisión o estrategia de negocio.

Programador: Si. Diseñador: ¡Ni de riesgos!

IT & SOCIEDAD

La jerga popular dice: "zapatero a tus zapatos", pero algunos gerentes de software olvidan la necesidad de tener roles especializados y lo que este pequeño descuido puede causar en la ejecución de un proyecto.

Escrito por: **Eliana Caraballo** (Ing. de sistemas)

Ingeniera de desarrollo senior y **miembro activo de AVANET**. Apasionada por todo lo relacionado con la ingeniería de software y los procesos de calidad para desarrollo. Siempre abierta a aprender y a compartir lo que los años de desarrollo me han enseñado.

Redes sociales:
Twitter / Identi.ca: **@elianaca**
LinkedIn: http://co.linkedin.com/in/elianacaraballoa

Hace un tiempo atrás tuve que enfrentarme a la peor pesadilla de casi todos los programadores que conozco: Tener que ajustar un CSS y organizar visualmente la plantilla acorde a un mockup entregado. Mi primera reacción fue avisarle a mi jefe que mis habilidades estéticas se resumían a preguntarle a mi hija si la ropa que tenía puesta combinaba y por más de que rogué que no me asignara esa tarea, igual tuve que hacerla "de la mejor manera posible".

¿Por qué se nos hará tan complicada la tarea de hacer una interfaz medianamente presentable? Para mí la respuesta es sencilla: somos programadores, no diseñadores gráficos. De las cosas que recuerdo que me enseñaron en la universidad es que la pantalla fluye de arriba hacia abajo y de izquierda a derecha; nada más. Eso de poner colores que combinen y que sea visualmente agradable es muchas veces una tarea titánica para la que no fuimos formados. Todos soñamos con llegar a un proyecto donde el diseñador gráfico ya hizo las plantillas y nosotros simplemente nos dedicamos a colocar los controles y a hacer nuestra magia.

Es de conocimiento "popular" que los ingenieros de sistemas solemos saber "de todo un poco", pero esto hace referencia específicamente al know how del negocio específico

para el que estamos desarrollando software: si no conocemos cómo funcionan las cosas actualmente muy posiblemente el producto final será defectuoso o "inútil" para el cliente. Otra cosa muy diferente, es que se pretenda que cubramos todos los roles de un proyecto. Son muchas las anécdotas acerca de ser los "toderos", ya sea por falta de presupuesto o porque el proyecto es tan pequeño que no amerita tanto personal involucrado, pero es acá donde se encontrarán la mayoría falencias: en las áreas donde somos menos fuertes.

A lo largo de los proyectos donde he estado, he notado que cuando a un programador se le asigna una tarea que no es de su competencia, o sin la suficiente preparación para la misma, los resultados que se obtienen son medianamente aceptables y por lo general necesitan re-proceso en un determinado momento. Algunos de los casos más comunes son: cuando nos ponen a hacer el modelo de la base de datos y nuestra experiencia con ellas no va más allá de lo visto en la Universidad, donde la normalización, nombrado, relaciones y demás tendrán grandes vacíos y a veces terminan en un salpicón de retazos. Cuando nos ponen a diseñar una arquitectura y a duras penas hemos leído al respecto, obteniendo como resultado patrones de desarrollo desajustados o el uso de capas innecesarias, volviéndose un código complicado de mantener. Así hay muchas más malas experiencias en desarrollo, en donde se hace evidente que si no se tienen claras las competencias de la persona encargada de la tarea, se puede incurrir en re-procesos y/o sobrecostos que pudieron ser evitados desde el comienzo.

En muchas casas de software, especialmente las pequeñas y más nuevas, la falta de presupuesto y/o personal especializado para cada uno de los roles, haciendo que estas personas que no están preparadas generen un producto de calidad "aceptable". Ya sé, muchos me dirán que el cliente pondrá peros por el sobrecosto del proyecto al pagar personas "adicionales", pero es importante que tengan en cuenta que a medida que van creciendo, se hace necesario definir los roles y tener el personal capacitado para el mismo. Una buena opción que he visto en varias empresas es que escogen a los junior y los van encaminando por el área donde muestran fortalezas, de modo que van haciendo "carrera" y al cabo de un tiempo se puede llegar a tener el personal idóneo para cada rol de un proyecto.

La especialización de las tareas es algo fundamental en el desarrollo de software desde mi punto de vista, pues permite a las personas estar concentradas en hacer bien lo que realmente saben, en lugar de tratar de aprender de manera express y vía Google™ acerca de áreas donde no son fuertes. Un buen gerente es capaz de identificar y potenciar las aptitudes de las personas que están en su equipo, de modo que se sientan motivados, útiles, y sobretodo, haciendo cosas para las que son realmente buenos. Si alguien del equipo muestra interés en alguna otra área, es importante poder propiciar el espacio para que pueda aprender y complementar sus conocimientos, porque tampoco es propicio que se vaya sintiendo "estancada" en determinado proceso.

Mi invitación es a las empresas a tener siempre en cuenta los roles definidos para cada una de las tareas del proyecto y asignarle el personal idóneo a las mismas para garantizar su éxito en la ejecución. Si por alguna razón esto no es posible, sea por falta de personas con las competencias necesarias o alguna otra razón, tenga presente que debe dar un tiempo prudencial para que aprenda lo suficiente sobre esta nueva competencia y especialmente, no espere que el trabajo sea excelente al primer intento, porque probablemente recibirá la crítica que recibió mi jefe cuando el cliente vio las

plantillas que yo había hecho: “Esto fue hecho por un niño de preescolar”, lo que hizo que terminara mi corta carrera en el diseño gráfico.

PSeInt: Una Invitación para entrar en el maravilloso mundo de la programación

PROGRAMACIÓN – PSEUDO CÓDIGO

Muchas personas que recién inician en el arte de programar se preguntan como y por donde empezar, si el lenguaje que escogieron estará bien o mal y erradamente muchas veces se preocupan más por aprender una herramienta que por la lógica de programación (que es lo realmente importante), veremos como PSeInt es una gran ayuda al iniciar en este maravilloso mundo y cuales son algunos puntos esenciales a tomar en cuenta si quieres empezar a “codear”.

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

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

Webs:
Blog: www.milale.net

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

Al entrar en el maravilloso mundo de la programación las personas suelen tener ciertos miedos, sobretodo cuando piensan si el lenguaje con el que se inician será el más adecuado, o que hubiera pasado si escogían otro, pero sin considerar lo más importante que es aprender la lógica de programación ya que así se les hará más fácil usar cualquier herramienta al ya tener las nociones de como "codear".

La lógica de programación es la base sobre la cual se sustenta la programación, cuando alguien empieza a codear debe enfrentarse a dos grandes preguntas: **Qué** procesos realizar para resolver el problema y el **cómo** escribir el código (procesos del **qué**).

El pseudocódigo es una descripción de un algoritmo de programación que combina las convenciones de un lenguaje verdadero y un idioma (español, ingles, etc) ya que está diseñado para la lectura humana sin considerar código específico, es más fácil de entender que el código convencional ya que describe los procesos que un programa contendrá de manera sencilla.

Existen varias herramientas que pueden ayudar a aprender la lógica de programación, en particular uno de los IDE que me gusta es PSeInt (Pseudo Intérprete), está bajo licencia pública general de GNU, orientado a personas que recién empiezan a programar, su éxito es tal que a fecha 15 de Noviembre del 2012 figura dentro de los 20 proyectos con mayor número de descargas en SourceForge; para obtenerlo solo necesitas entrar a la página oficial[0] y en la sección Descargas[1] se encuentra toda la documentación necesaria para la instalación.

> *Cualquier tonto puede escribir código que un ordenador entiende. Los buenos programadores escriben código que los humanos pueden ententer". Martin Fowler*

¿Porqué PSeInt?

PSeInt es una herramienta que ayuda a entender más sobre el pseudocódigo ya que presenta herramientas de edición como autocompletado, ayudas, plantillas, coloreado de sintaxis, indentado inteligente, generación de diagramas de flujo, ejecución de programas escritos paso a paso detallando procesos de prueba y errores; a parte de todo esto el plus es que es multiplataforma y totalmente libre y gratuito.

Algoritmos en Pseudocódigo

Generalmente todo algoritmo escrito en pseudocódigo sigue una estructura: el uso de palabras claves (Proceso, FinProceso, Si, Sino, FinSi, Para, FinPara, etc) y una secuencia de instrucciones. Aquí tenemos un ejemplo básico para una suma de números e identificar si este resultado es par o impar:

```
Proceso SumaNumeros_ParImpar //Cabecera del algoritmo

      //Cuerpo del algoritmo
      Escribir "Introduzca primer numero:"; //Escribir muestra valores
      Leer numero01; //Leer ingresa información

      Escribir "Introduzca segundo numero:";
      Leer numero02;

      suma<-numero1 + numero2; //Asignación (almacena valor en la variable)

      Escribir "La suma es: ",suma;

      //Condicional Si para saber si es par o impar

      Si suma % 2 = 0 Entonces
            Escribir "La suma es un valor par";
      Sino
            Escribir "La suma es un valor impar";
      FinSi

FinProceso
```

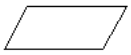
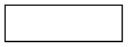
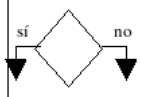
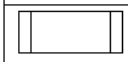
Y así de una manera sencilla podemos hacer muchas más acciones que el pseudocódigo nos permite, como lazos de repetir, mientras, para, etc de acuerdo el problema que deseemos resolver; los operadores usados son los siguientes:

Operador	*Significado*	*Ejemplo*
Relacionales		
>	Mayor que	3>2
<	Menor que	'ABC'<'abc'
=	Igual que	4=3
<=	Menor o igual que	'a'<='b'
>=	Mayor o igual que	4>=5
Logicos		
& ó Y	Conjunción (y).	(7>4) & (2=1) //falso
| ó O	Disyunción (o).	(1=1 | 2=1) //verdadero
~ ó NO	Negación (no).	~(2<5) //falso
Algebraicos		
+	Suma	total <- cant1 + cant2
-	Resta	stock <- disp - venta
*	Multiplicación	area <- base * altura
/	División	porc <- 100 * parte / total
^	Potenciación	sup <- 3.41 * radio ^ 2
% ó MOD	Módulo (resto de la división entera)	resto <- num MOD div

Diagramas de flujo

Es la representación gráfica del algoritmo o proceso, en PSeInt también podremos obtener el diagrama de nuestro algoritmo en pseudocódigo, se trata de decir lo mismo

pero de una manera gráfica y más entendible ya que permite visualizar el camino que sigue la solución del problema, aquí podemos ver los símbolos usados en este tipo de diagramas.

Símbolo	Función
	**Terminal.** Representa el comienzo o el fin de un programa.
	**Entrada** / **Salida.** Indica una introducción de datos desde un dispositivo externo (por defecto, el teclado) o una salida de datos hacia algún dispositivo externo (por defecto, la pantalla)
	**Proceso.** Representa cualquier operación que se lleve a cabo con los datos del problema.
	**Condición**. Señala una bifurcación del flujo de instrucciones. La bifurcación está siempre controlada por una operación relacional llamada *condición*, cuyo resultado puede ser "verdadero" o "falso" (o también "sí" o "no"), dependiendo del valor de los datos de la expresión condicional. En función del resultado de dicha expresión, el flujo de ejecución continúa por una u otra rama (pero nunca por las dos a la vez)
	**Condición múltiple**. Sirve para indicar una bifurcación del flujo en varias ramas, no sólo en una. En este caso, la condición no puede ser booleana, sino entera.
	**Conector.** Para enlazar un fragmento del diagrama de flujo con otro fragmento situado en la misma página. Se usa cuando el diagrama es muy grande y no puede dibujarse entero de arriba a abajo.
	**Conector**. Como el anterior, pero para conectar un fragmento del diagrama con otro fragmento situado en una página diferente.
	**Dirección del flujo**. Indica el orden de ejecución de los pasos del algoritmo.
	**Subrutina**. Llamada a un subproceso o módulo independiente (ver apartado de "Programación Modular")

PSeInt ofrece muchas funcionalidades[2] y de esta manera te permitirá realizar ejemplos [3] claros y sencillos para comprender todo lo que necesitas saber desde el nivel básico y convertirte en un muy buen programador para tener un “happy hacking”.

“Code is poetry”

Si estas empezando o quieres empezar en este arte (de la programación) es importante que primero le des mucha importancia a la lógica, siendo coherente con tus ideas y razonamientos; y luego busques el lenguaje indicado para ti. Necesitas de paciencia (mucha) para poder buscar una solución a los problemas que desees resolver y piensa en lo gratificante que será luego cuando lo hayas conseguido, la perseverancia es muy importante para poder combatir los bloqueos, desconocimientos y demás que al final nos llevarán a ser buenos en lo que hacemos; debemos pensar en todas las posibles soluciones ya que mientras de más puntos o lados lo miremos aumentamos la posibilidad de obtener la mejor solución. Recuerdo cuando empezaba y me dieron un gran consejo, el aprender practicando (no hay mejor forma que esa). Verás que después de un tiempo disfrutarás mucho de “quedarte pegado” horas de horas en el código hasta que funcione.

01101000 01100001 01110000 01110000 01111001 0100000 01100011 01101111 01100100 01101001 01101110 01100111!!!

Referencias & Links

[0] PSeInt – Web oficial [en línea]: <http://pseint.sourceforge.net/> [Consulta: 18 de Noviembre de 2012]

[1] PSeInt - Descargas [en línea]: <http://pseint.sourceforge.net/index.php?page=descargas.php> [Consulta: 18 de Noviembre de 2012]

[2] PSeInt – Videos e imágenes [En línea]: <http://pseint.sourceforge.net/index.php?page=imagenes.php> [Consulta: 18 de Noviembre de 2012]

[3] Ejemplo usando PSeInt [En línea]: <http://blip.tv/neosergio/programa-piloto-fundamentos-de-programaci%C3%B3n-4162613> [Consulta: 21 de Noviembre de 2012]

¿Qué son los Namespaces?

PHP

Desde que empezó esta casi revolución de creación de librerías, plugins y uso de componentes de terceros, que definitivamente nos ayudan a mitigar todo ese tiempo de desarrollo -casi siempre-, con ello nos vino el gran problema: los nombres de clases, constantes, funciones. Y si nuestra creatividad no nos daba para más y terminamos visionando un nombre como «user» o «bd» y ya había sido declarado, nos topábamos con un «name collision». Para esto PHP 5.3 nos trajo una solución: «namespaces.

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

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

Webs:
About.me: http://about.me/indirabm

Redes sociales:
Twitter: **@indirabm**

Ahora, hablando de realidades, los namespaces no son cosa nueva. Ya los tenían implementados Java, Ruby o Python. Hablemos un poco de historia: Antes de los namespaces, la única solución conocida era tener class/function, es así que por ejemplo con Zend tenías nombres de clases como este:

```
Zend_CodeGenerator_Php_Docblock_Tag_License
```

Como pueden darse cuenta los nombres de clases terminaban siendo muy grandes, lo que nos dejaba una estructura poco organizada y limpia y por supuesto, a más grande tu proyecto, el nombre podría terminar siendo una verdadera oración.

> *"Es una manera de encapsular elementos. Se puede ver como un concepto abstracto en muchos aspectos. Por ejemplo, en cualquier sistema operativo los directorios sirven para agrupar archivos relacionados, y actúan como espacios de nombres para los archivos que hay en ellos."*[2]

Demos los primeros pasos

Para un mejor entendimiento reproduzcamos un *name collision*.

```
/*
 * Este es un requerimiento de compra de un artículo de oficina.
 */
class requerimiento() {
      private $descripcion = "Papeles de 5cm x2mm";

      public function getDesc(){
            return $this->descripcion;
      }

      public function setDesc($desc){
            $this->descripcion = $desc;
      }
}


/*
 * Este es un requerimiento de documentos a llevar para ingresar a un
 * concierto.
 */
class requerimiento() {
      private $descripcion = "Llevar documento de identidad";

      public function getDesc(){
            return $this->descripcion;
      }
```

2 http://www.php.net/manual/es/language.namespaces.php

```
        public function setDesc($desc){
              $this->descripcion = $desc;
        }

  }
```

Estos son dos conceptos distintos y para poder separarlos tenemos que definir su namespace, así que, manos a la obra:

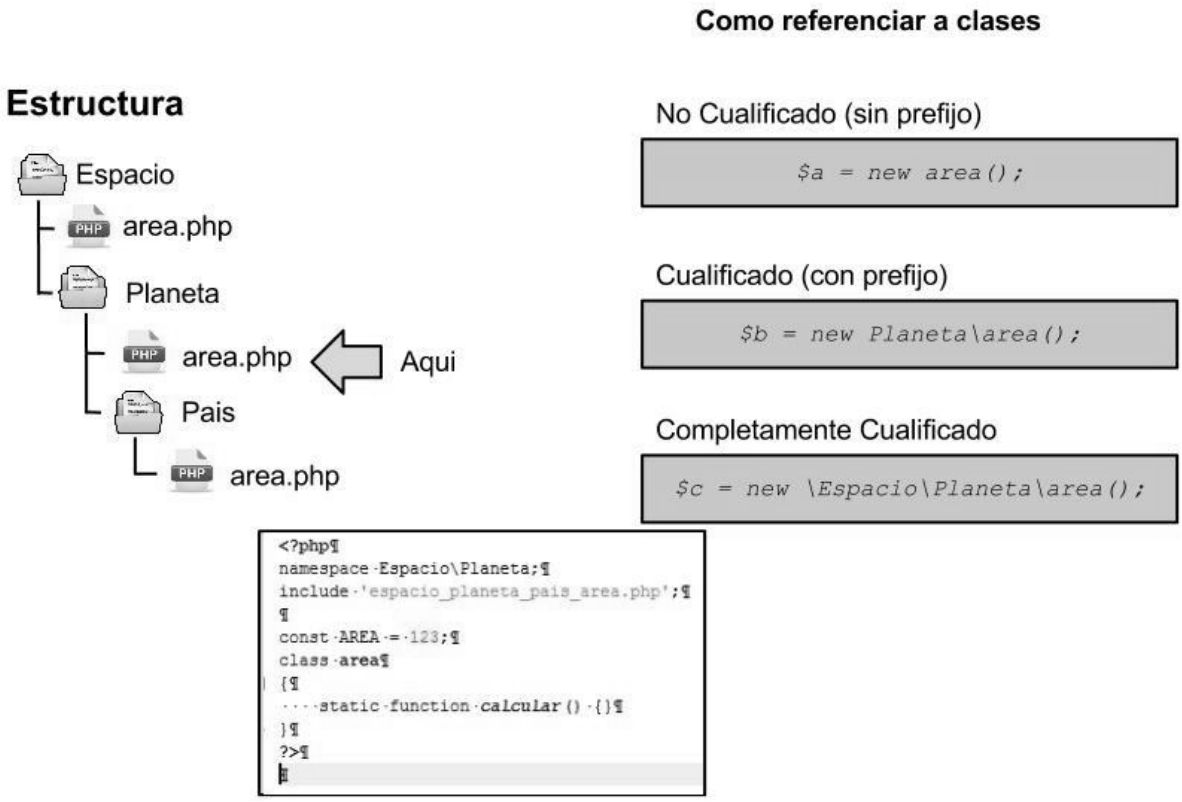
```
  namespace Compras;
  class Requerimiento() {

        private $descripcion = "Papeles de 5cm x2mm";

        public function getDesc(){
              return $this->descripcion;
        }

        public function setDesc($desc){
              $this->descripcion = $desc;
        }
  }


  namespace Concierto;
  class Requerimiento() {

        private $descripcion = "Llevar documento de identidad";

        public function getDesc(){
              return $this->descripcion;
        }
        public function setDesc($desc){
              $this->descripcion = $desc;
        }
  }
```

Es así que hemos podido definir dos *namespaces*. Es importante aclarar que para efectos de ejemplo se han puesto dos *namespace* en un mismo documento.

> *No es recomendable definir mas de un namespace en un mismo archivo.*

¿Como entenderlo?

Aquí agregaremos un ejemplo para explicar el funcionamiento de los *namespaces* y los 3 tipos de referencias que existen:



Como podrán ver en la imagen, en la parte izquierda tenemos la estructura de un programa para medir el área. Por supuesto, hay un área para espacio, planeta y país. Sin embargo, nosotros nos ubicamos en la flecha *(NdR: Flecha con la leyenda "Aquí")*, en la clase área del planeta. Desde aquí, veremos como se hace referencia a un nombre de clase que esta dado por 3 formas diferentes:

No cualificado

Como ven, éste no tiene prefijo y en el código que se puede ver en la parte media de la imagen, ya cuenta con su namespace.

Cualificado

Esta es otra forma sin embargo se usa un prefijo.

Completamente cualificado

Este utiliza el operador de prefijo global.

¿Como usarlo?

Ahora mostraremos 3 diferentes formas de invocar a un namespace. Digamos que para acceder necesitamos un controlador. Entonces llamaremos al namespace del controlador. Veamos las diferentes formas de invocar.

(1) Como vemos, se ha prefijado el nombre de la clase con el namespace. Además, tener en cuenta que el namespace debe ser declarado al inicio del archivo, antes que cualquier

otro código, con excepción de la palabra clave declare.

```
namespace Controller;
require_once 'compras/Requerimiento.php';

$req = new \Compras\Requerimiento();
var_dump($req->getDesc());
```

(2) Una característica muy importante de los namespaces es el de importar, esto es similar a la capacidad de los sistemas de archivos basados en *nix, de crear enlaces simbólicos a un archivo o directorio[3]. Para ello usamos la palabra **use**. Esto nos ayudará cuando tengamos muchos objetos ubicados en un mismo namespace.

```
namespace Controller;
require_once "compras/Requerimiento.php";
use \compras\Requerimiento;

$req = new Requerimiento();
var_dump($req->getDesc())
```

(3) Esta también cuenta con un **alias**, que nos servirá para cuando en un archivo existan varios niveles de namespaces.

```
namespace Controller;
require_once "compras/Requerimiento.php";
use \compras\Requerimiento as Reque;

$req = new Reque();
var_dump($req->getDesc())
```

Algunos tips:

- Los namespace se pueden definir con subniveles, por ejemplo:

  ```
  namespace Proyecto\Nivel\SubNivel;
  ```

- Una buena práctica con las namespaces de PHP para cualificarlos en el código, es usar referencias absolutas, en lugar de relativas.
- Ni las funciones ni las constantes se pueden importar mediante la sentencia **use**.
- Hay que tener cuidado con los nombres a los namespaces dado que no se pueden usar palabras reservadas

  ```
  namespace Project/Abstract/Factory
  ```

Después de haber aprendido sobre los namespaces, hagamos un resumen de sus beneficios:

- Empaquetar diferentes funcionalidades o librerías que puedan tener clases con nombres redundantes, lo hace mas modular.

3 http://www.php.net/manual/es/language.namespaces.php

- Posibilidad de usar un namespace definido con la palabra reservada **use**, para funciones en un mismo namespace, además de la utilización del **alias**.
- Facilitar la dominación y la identificación de clases.

Ahora ya sabemos qué es un namespace. Démosle una ojeada a algunos de los frameworks que ya lo tienen implementado.

- **Symfony 2[4]:**

  Este puede ser uno, o uno de los más antiguos frameworks php, que implementó el uso de namespaces.

- **Zend Framework 2.0[5]:**

  Este "Goliat" de las librerías en su versión 2.0 ya cuenta con namespaces.

- **YII Framework[6]:**

  Tiene implementado los namespaces para clases, controladores y módulos.

4 http://symfony.com/
5 http://framework.zend.com/
6 http://www.yiiframework.com/

Manual de MVC: (1) FrontController

ARQUITECTURA

Desde esta segunda edición de Hackers & Developers Magazine, damos inicio al Manual de MVC en Python y PHP. Cada mes, una nueva entrega del manual, abarcando en detalle, el desarrollo de aplicaciones Web modulares con el patrón arquitectónico modelo-vista-controlador. En esta primera entrega, haremos una introducción a MVC y hablaremos sobre el FrontController: el patrón de diseño, que nos permitirá utilizar una arquitectura MVC en sistemas modulares.

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

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

Webs:
www.cursosdeprogramacionadistancia.com
www.eugeniabahit.com

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

FrontController es el **patrón de diseño** requerido para trabajar con **MVC**, en **arquitecturas modulares**, ya que el mismo, permite manejar desde un único archivo, las peticiones del usuario, analizarlas e invocar al controlador responsable de proveer la información solicitada al usuario. Pero antes de ir de lleno al `FrontController`, haremos una breve introducción sobre el Patrón Arquitectónico (mal llamado "patrón de de diseño"), MVC.

Entendiendo el Patrón Arquitectónico modelo-vista-controlador

El patrón MVC debe ser entendido desde dos ópticas diferentes: la del desarrollador y la

del usuario. Desde el punto de vista del desarrollador, MVC nos permite mantener individualizadas, las "responsabilidades" dentro de un sistema, permitiéndonos diferenciar y aislar el diseño del sistema (objetos que componen los "modelos"), de la interfaz gráfica del usuario (GUI) y su correspondiente lógica de negocios (vistas), utilizando como "conector intermediario" un objeto controlador.

El orden modelo-vista-controlador, es el que el programador debe seguir en el proceso de desarrollo del sistema. Sin embargo, desde el punto de vista del usuario, todo comienza en el controlador. Veamos cómo:

- A nivel funcional, en MVC todo se inicia con una solicitud del usuario (petición);
- Dicha solicitud (algo que el usuario quiere hacer con respecto al sistema), es representada mediante la estructura de la URI;
- En una arquitectura modular, la estructura de la URI, guardará la forma: (dominio)/modulo/modelo/recurso[/argumentos] siendo modulo, modelo y recurso datos obligatorios y argumentos, opcionales. De esta forma, si un usuario quisiera agregar una nueva nota de crédito y el modelo (objeto) "comprobante", perteneciese al módulo de contabilidad, dicha petición sería realiza a través de la siguiente URI: http://mymvcapp.net/contabilidad/comprobante/agregar
- Cuando la solicitud del usuario es enviada, ésta es recibida por el controlador del modelo, quien se comunica con éste solicitándole la información necesaria;
- Una vez que el modelo retorna la información al controlador, éste le entrega dicha información a la vista del modelo, quien será la encargada de procesar la información recibida, colocarla en la GUI y mostrársela al usuario.

Veamos un ejemplo:

```
# MODELO: /myapp/modulo/models/vidrio.php
class Vidrio {

    function __construct() {
        $this->vidrio_id = 0;
        $this->color = '';
    }

    function save() {
        # Guarda un nuevo objeto u objeto existente
    }

    function get() {
        # Recupera un objeto
    }

    function destroy() {
        # Destruye un objeto
    }

}
```

```
# MODELO: /myapp/modulo/models/vidrio.py
class Vidrio(object):

    def __init__(self):
        self.vidrio_id = 0
        self.color = ''

    def save(self):
        """Guarda un nuevo objeto u objeto existente"""
        pass

    def get(self):
        """Recupera un objeto"""
        pass

    def destroy(self):
        """Destruye un objeto"""
        pass


# GUI (para PHP): /myapp/static/html/ver_vidrio.html
<h1>Vidrio {vidrio_id}</h1>
<p>Vidrio de color {color}.</p>


# VISTA: /myapp/modulo/views/vidrio.php
class VidrioView {

    function ver($objeto=NULL) {
        settype($objeto, 'array');
        $comodines = array_keys($objeto);
        foreach($comodines as &$comodin) {
            $comodin = "\{$comodin\}";
        }
        $valores = array_values($objeto);
        $template = file_get_contents("/myapp/static/html/ver_vidrio.html");
        print str_replace($comodines, $valores, $template);
    }

}



# GUI (para Python): /myapp/static/html/ver_vidrio.html
<h1>Vidrio $vidrio_id</h1>
<p>Vidrio de color $color.</p>

# VISTA: /myapp/modulo/views/vidrio.py
class VidrioView(object):

    def ver(self, objeto):
        with open("/myapp/static/html/ver_vidrio.html", "r") as archivo:
            template = archivo.read()
        return Template(template).safe_substitute(vars(objeto))
```

```
# CONTROLADOR: /myapp/modulo/controllers/vidrio.php
class VidrioController {

    function ver($id=0) {
        $vidrio = new Vidrio();
        $vidrio->vidrio_id = $id;
        $vidrio->get();

        $view = new VidrioView();
        $view->ver($vidrio);
    }

}
```

```
# CONTROLADOR: /myapp/modulo/controllers/vidrio.py
class VidrioController(object):

    def ver(self, id=0):
        vidrio = Vidrio()
        vidrio.vidrio_id = id
        vidrio.get()

        view = VidrioView()
        self.output = view.ver(vidrio)
```

Si el usuario quisiera ver el vidrio con id 15, su solicitud sería enviada a través de: http://mymvcapp.net/modulo/vidrio/ver/15 y la misma, sería tramitada por VidrioController, pero **¿cómo llegará la solicitud a VidrioController?** Para responder a esta pregunta, **tendremos que hablar de FrontController**.

Recibiendo las solicitudes del usuario con FrontController desde Python

En Python (bajo el supuesto que trabajamos con WSGI bajo Apache[7]) nuestra `application` será quien haga las veces de FrontController. En primer lugar, deberá analizar la URI a fin de obtener el módulo, modelo, recurso y los argumentos opcionales que le permitirán conocer a qué controlador debe invocar:

```
# Divido la URI utilizando como separador la barra diagonal
peticiones = environ['REQUEST_URI'].split('/')

# Elimino el primer elemento puesto que estará vacío
peticiones.pop(0)

# Cuento las peticiones, para saber si hay o no argumentos
```

7 Si no sabes como codear una Web en Python "crudo" (sin Frameworks) corriendo bajo Apache, te recomiendo leer un artículo que publiqué en Debian Hackers **"Una Web en Python sobre Apache en 3 pasos"** ingresando en http://www.debianhackers.net/una-web-en-python-sobre-apache-sin-frameworks-y-en-solo-3-pasos

```
cantidad = len(peticiones)

# Obtengo el módulo, modelo, recurso (y argumentos, si existen)
if cantidad == 3:
    modulo, modelo, recurso = peticiones
elif cantidad == 4:
    modulo, modelo, recurso, arg = peticiones
```

A continuación, deberá -utilizando los datos obtenidos-, importar el módulo del controlador, instanciarlo y entregarle la información necesaria para que éste actúe:

```
# Obtengo el nombre del controlador
controller_name = '%sController' % modelo

# Para poder importar el controlador, debo agregar el path de la aplicación
from sys import path
path.append(environ['SCRIPT_FILENAME'].replace('frontcontroller.py', ''))

# Importo el módulo del controlador
exec 'from %s.controllers.%s import %s' % (modulo, modelo, controller_name)

# Instancio al controlador y le envío el recurso y argumentos
controller = locals()[controller_name](recurso, arg)

# Capturo la salida
output = controller.output
```

Finalmente, se llama a start_response() y se retorna la salida, normalmente:

```
def application(environ, start_response):
    peticiones = environ['REQUEST_URI'].split('/')
    peticiones.pop(0)
    cantidad = len(peticiones)

    if cantidad == 3:
        modulo, modelo, recurso = peticiones
    elif cantidad == 4:
        modulo, modelo, recurso, arg = peticiones

    controller_name = '%sController' % modelo.capitalize()

    from sys import path
    path.append(environ['SCRIPT_FILENAME'].replace('frontcontroller.py', ''))

    exec 'from %s.controllers.%s import %s' % (modulo, modelo,
        controller_name)

    controller = locals()[controller_name](recurso, arg)
    output = controller.output

    start_response('200 OK', [('Content-Type', 'text/html; charset=utf-8')])
    return output
```

El FrontController, nos obligará a que nuestros controladores, estén preparados para

recibir dos parámetros: el recurso (para hacer la llamada dinámica al método actuante) y el argumento (que pueda ser requerido por algunos métodos):

```
class VidrioController(object):

    def __init__(self, recurso='', arg=0):
        getattr(self, recurso)(int(arg))

    def ver(self, id=0):
        vidrio = Vidrio()
        vidrio.vidrio_id = id
        vidrio.get()

        view = VidrioView()
        self.output = view.ver(vidrio)
```

Recibiendo las solicitudes del usuario con FrontController desde PHP

En PHP, el trabajo que debemos dejar preparado para poder actuar, requiere un mayor esfuerzo. En primer lugar, debemos preparar a nuestro servidor, para trabajar con las denominadas "Friendly URL" (URL amigables). Esto requiere:

Habilitar el módulo `rewrite` de Apache (debes convertirte en súper usuario):

```
a2enmod rewrite
```

Modificar la directiva AllowOverride de nuestro VirtualHost:

```
AllowOverride All
```

(Si utilizas el `VirtualHost` por defecto, localizarás el archivo en `/etc/apache2/sites-available/default`. Ten la precaución de modificarlo como súper usuario. La directiva `AllowOverride` se encuentra dentro del tag `Directory`) ¡No olvides **reiniciar Apache** para que los cambios surjan efecto! `service apache2 restart`

Y finalmente, crear un archivo `.htaccess` en el directorio raíz de la aplicación (estará en el mismo directorio que frontcontroller.php)

```
# Encender el motor de reescritura de las URL
RewriteEngine On
# Crear una regla que redirija todas las peticiones (excepto las realizadas
# al directorio static) hacia el frontcontroller.php
RewriteRule !(^static) frontcontroller.php
```

Una vez "preparado el terreno", estaremos en condiciones de crear nuestro FrontController.

Al igual que en Python, el primer paso será analizar la URI a fin de obtener el módulo, el modelo, el recurso y los argumentos que opcionalmente, podría estar enviando el

usuario:

```
# Divido la URI utilizando como separador la barra diagonal
$peticiones = explode('/', $_SERVER['REQUEST_URI']);

# Cuento las peticiones, para saber si hay o no argumentos
$cantidad = count($peticiones);

# Obtengo el módulo, modelo, recurso (y argumentos, si existen)
if($cantidad == 3) {
    list($modulo, $modelo, $recurso) = $peticiones;
} elseif($cantidad == 4) {
    list($modulo, $modelo, $recurso, $arg) = $peticiones;
}
```

Luego, con los datos obtenidos, está en condiciones de importar el archivo del controlador , instanciarlo y entregarle la información necesaria para que éste actúe:

```
# Obtengo el nombre del controlador
$controller_name = ucwords($modelo) . "Controller";

# Para poder importar el controlador, debo agregar el path de la aplicación
ini_set('include_path', str_replace('frontcontroller.php', '',
    $_SERVER['SCRIPT_FILENAME']));

# Importo el módulo del controlador
require_once("$modulo/controllers/$modelo.php");

# Instancio al controlador y le envío el recurso y argumentos
$controller = new $controller_name($recurso, $arg);
```

Finalmente, el FrontController deberá auto-ejecutarse:

```
class FrontController {

    public static function handler() {
        $peticiones = explode('/', $_SERVER['REQUEST_URI']);
        $cantidad = count($peticiones);
        if($cantidad == 3) {
            list($modulo, $modelo, $recurso) = $peticiones;
        } elseif($cantidad == 4) {
            list($modulo, $modelo, $recurso, $arg) = $peticiones;
        }
        $controller_name = ucwords($modelo) . "Controller";
        ini_set('include_path', str_replace(
            'frontcontroller.php', '', $_SERVER['SCRIPT_FILENAME']));
        require_once("$modulo/controllers/$modelo.php");
        $controller = new $controller_name($recurso, $arg);
    }

}

FrontController::handler();
```

> *Por favor, notar que la directiva ini_set, debería estar en un archivo de configuración (un settings.php) y no, dentro del FrontController.*

Nuestro FrontController, ahora nos estará obligando a preparar los controladores de nuestros módulos, para recibir dos parámetros: el recurso (para hacer la llamada dinámica al método actuante) y el argumento (que pueda ser requerido por algunos métodos):

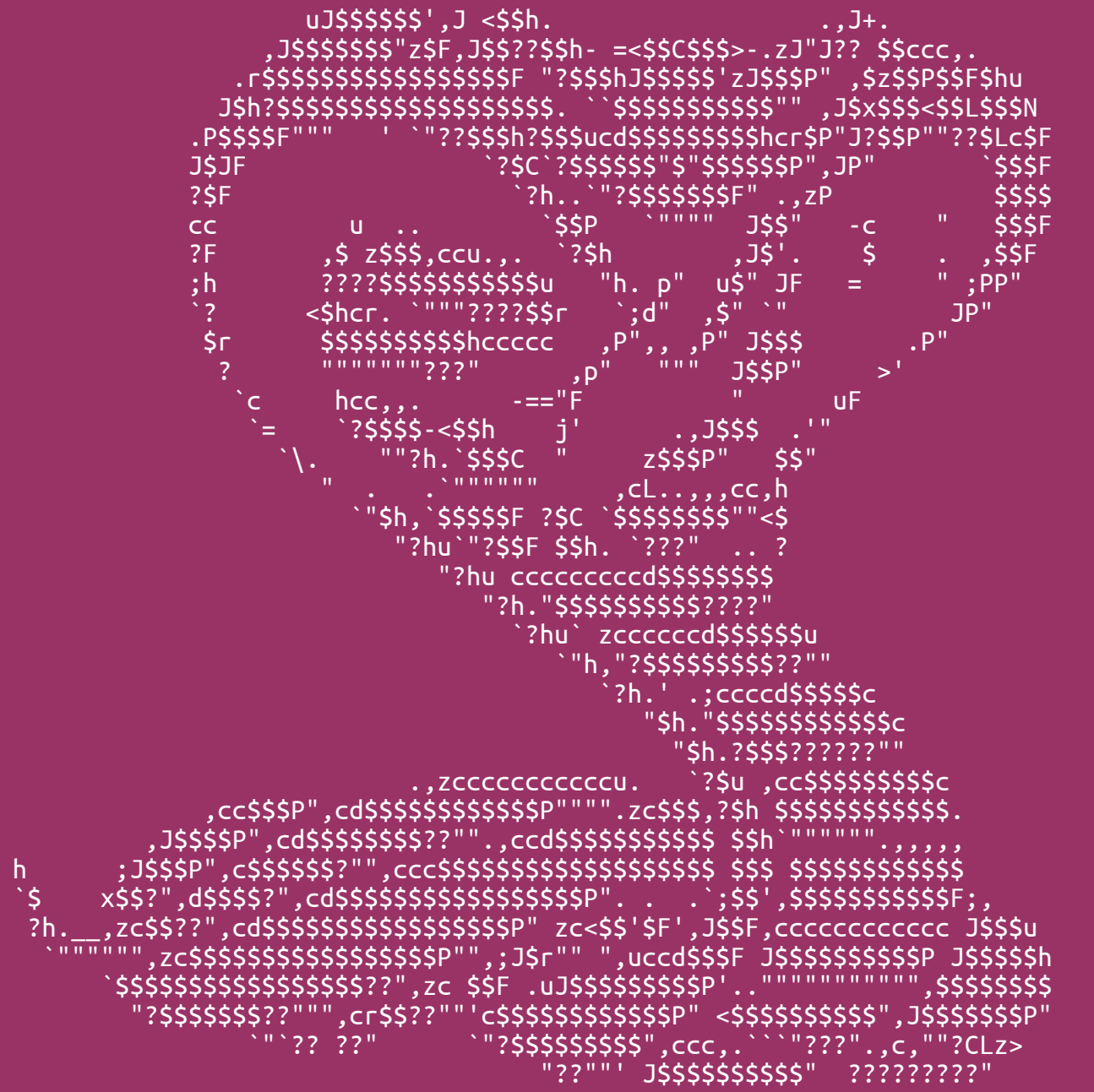
```
class VidrioController {

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

    function ver($id=0) {
        $vidrio = new Vidrio();
        $vidrio->vidrio_id = $id;
        $vidrio->get();

        $view = new VidrioView();
        $view->ver($vidrio);
    }

}
```

En la siguiente entrega de MVC en Python y PHP, nos ocuparemos especialmente de las vistas.

ASCII ART

Cobra

por **Anónimo**



TU ZONA EXCLUSIVA

U!

Para **publicar** en la zona **U!** envíanos tu mensaje a **contacto@hdmagazine.org** o a través de Twitter incluyendo el hashtag oficial **#HDMagazine**.

Los 3 deseos cumplidos por Hackladino:

Víctor (Lima Perú): (...) Me preguntaba si podrían hacer un artículo sobre el "modelo vista controlador" orientado a la programación web (PHP y POO) . **Ariel Scherman (Bs. As. Argentina):** (...) Estaría bueno algún artículo sobre MVC. **@cruzrovira:** (...) algo de geo-localizacion.

HD Responde...

Sergio Lopez de Abaria: Igual que hablais del framework para php, me gustaria comentarais para distintos lenguajes de programacion, podria ser como una serie, cada semana uno, por ejemplo html5 y css3, pero que los valoreis vosotras, el mas popular, el mas facil o intuitivo, el mejor, etc, gracias

Respuesta: *es muy buena idea! Por el momento, los temas de la revista están enfocados al hacking y la programación en relación al Software Libre y aún no hemos contemplado expandirnos hacia el diseño Web. Motivo por el cual, no contamos con columnista para temas de diseño. Pero confiamos en poder extender la temática del Magazine, también hacia el diseño Web, dentro de poco tiempo.*

Vicente (desde Alicante, España): (...) me gustaría que me enviaseis algún dato más o bibliografía a la que poder acudir para enseñarme la programación en Python (...) por ultimo, que enfoque podría darle de salida para un posible trabajo gracias a este lenguaje de programación.

Respuesta: *Un muy buen material (libre) para Python, es el libro "Python para todos" de Raul Gonzalez Duque que puedes obtener en http://mundogeek.net/tutorial-python/). Con respecto a tu segunda consulta, hoy en día, Python es un lenguaje que está poniéndose "de moda". Compañías de la magnitud de Google™ y Canonical®, contratan profesionales tanto para desarrollos Web-based como desktop, mientras que otras más pequeñas, apuntan más a Frameworks como Django o Web2Py. Así que el mercado, en este sentido, es muy amplio =)*

Anónimo (visitante2011@.....): Escribo para felicitarlos por su revista y contenido tan interesante, quería también consultar si en cuanto a programación han considerado algo con Java para futuras publicaciones, ya que este lenguaje es muy utilizado también. Nuevamente felicidades y muchas gracias por su trabajo.

Respuesta: *nos encantaría incorporar material sobre desarrollo de Java en entornos GNU/Linux! Aún no contamos con columnista para este lenguaje, pero estamos en busca de profesionales que puedan hacer sus aportes. Esperamos contar pronto con una sección Java!*

Se comenta en Twitter...

@ralfrod: excelente primera edición: muy buenos artículos, buena diagramación y contenido variado. Felicitaciones!!

@vicmanfc: ¡Saludos a todo el equipo! Me ha encantado vuestra publicación, me encanta vuestra iniciativa. ¡Enhorabuena y ánimo!

@moonsmuse: Mi más sincera enhorabuena a todas por este proyecto. Seguid así. Acabo de descargarme el número 0 y ya soy fan vuestro.

@pronuer: un saludo a @indirabm y @eugeniabahit, felicitaciones por sus artículos de PHP escritos para @HackDevMagazine