AÑO ---------- 0
NÚMERO -------- 7
FECHA: 2013-05-27

#7

"SHE"

HD Hackers & DEVELOPERS

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

Staff

Eugenia Bahit	GLAMP Hacker & eXtreme Programmer
Indira Burga	Ingeniera de Sistemas
María José Montes Díaz	Técnica en Informática de Gestión
Milagros Infante Montero	Est. Ingeniería de Sistemas
Sergio Infante Montero	Ingeniero de Software

HD Hackers & DEVELOPERS #7

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

Ayúdanos a continuar con este proyecto

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

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

"Hacker es alguien que disfruta jugando con la inteligencia"
Richard Stallman
Free Software, Free Society
(Pág. 97), GNU Press 2010-2012

En esta edición:

Guía completa sobre el manejo de archivos por línea de comandos.....4
El mundo de la criptografía y la psicología de las contraseñas.....17
Conociendo a Vala.....23
Refactoring: avanzando con las soluciones.....27
Inception Deck en el Agilismo.....36
NgiNX instalado y... ¿ahora qué?.....40
El olvidado mundo de las variables en PHP.....49

Y LAS SECCIONES DE SIEMPRE:

ASCII Art..... Pág. 56
Este mes: Homenaje ASCII a Betty Boop

Zona U!..... Pág. 57
La comunidad de nuestros lectores y lectoras

Créditos

Hackers & Developers Magazine es posible gracias al compromiso de:

Responsable de Proyecto
Eugenia Bahit

Responsables de Comunicación

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

Staff

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

Indira Burga
Ingeniera de Sistemas
about.me/indirabm

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

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

Sergio Infante Montero
Ingeniero de Software
neosergio.net

Difusión

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

www.debianhackers.net

www.desarrolloweb.com

www.desdelinux.net

E-mail de Contacto:

contacto@hdmagazine.org

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

GNU/Linux para programadores:

Guía completa sobre el manejo de archivos por línea de comandos

La importancia de que un programador tenga la capacidad de moverse libremente por línea de comandos radica en algo tan simple como en recordar que en los entornos de producción (servidores) no existe un entorno gráfico. Es por ello, que aprender a manejar el Sistema Operativo por línea de comandos es indispensable para cualquier programador. En este artículo, nos centraremos en el manejo y manipulación de archivos.

Escrito por: **Eugenia Bahit** (GLAMP Hacker & eXtreme Programmer)

Eugenia es **Arquitecta de Software**, docente e 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**, **The Linux Foundation** y **Debian Hackers**. Creadora de python-printr, Europio Engine y colaboradora de Vim.

Webs:
Cursos de programación: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

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

Es muy probable que al leer la frase “manipulación de archivos” se piense en comandos como `mv`, `cp` y `rm`. Si bien son comandos que se utilizan para mover, copiar y eliminar archivos y directorios respectivamente, la manipulación de archivos no solo se limita a ellos. Por línea de comandos es necesario acceder a las mismas opciones que accedes por entorno gráfico. Desde descargar archivos de la Web, hasta comprimirlos, descomprimirlos, modificarlos o examinar sus propiedades.

En este artículo intentaremos abarcar la manipulación y manejo de archivos en el sentido más amplio yendo desde lo más básico a lo que puedan ser consideradas funciones un poco más complejas.

Lo más básico: moverse por el sistema de archivos

Ya conocerás al comando `cd` para moverte entre directorios. Sin embargo, me sucede a menudo con muchos alumnos, ver que olvidan los atajos y las buenas prácticas. Pero con `cd` se puede hacer mucho más que simplemente `cd /ruta/`. Incluso, `cd` no lo es todo. Veámoslo en detalle.

Conocer el directorio actual, te permite evitar escribir innecesarias rutas cuando deseas navegar entre directorios:

```
eugenia@cococha-gnucita:~$ pwd
/home/eugenia
```

En el caso anterior, suponiendo que tu usuario fuese `eugenia`, está claro que no necesitarás escribir `cd /home/eugenia/mi-otra-carpeta` para ingresar en esta última.

Auto-completar el nombre de archivos y directorios, reduce el riesgo de errores (y su consiguiente "volver a escribir") así como también, ayuda a la falta de memoria, pues es muy difícil recordar rutas completas. Tras escribir los primeros caracteres de un nombre se puede completar de forma automática, pulsando la tecla `tab` una vez. Suponiendo que deseas ingresar en una carpeta llamada Documentos (o crees recordar que su nombre comienza por "Do" pero desconoces como sigue):

```
eugenia@cococha-gnucita:~$ cd Do<TAB>

# Auto-completará la palabra como
eugenia@cococha-gnucita:~$ cd Documentos/
```

La instrucción **<TAB>** indica que la tecla TAB (tabulación) debe ser pulsada

Si no se auto-completa, es señal de que existe más de un archivo o directorio en el que su nombre coincide con los caracteres escritos. En ese caso, pulsas la tecla `tab` 2 veces consecutivas, para que en pantalla te muestre las coincidencias.

```
eugenia@cococha-gnucita:~$ cd D<TAB><TAB>
Descargas/  Documentos/
eugenia@cococha-gnucita:~$ cd D
```

RECUERDA: La barra diagonal / delante del nombre de una carpeta, indica que la misma se encuentra en la raíz del sistema de archivos.

Caminar hacia adelante en el sistema de archivos es tan simple como escribir `cd carpeta1/carpeta2/etcétera`, pero para **caminar hacia atrás** puede que te interesen algunos atajos:

```
# Retroceder un directorio
eugenia@cococha-gnucita:~/Documentos/documentacion/lisp$ cd ..
eugenia@cococha-gnucita:~/Documentos/documentacion$

# Retroceder varios directorios
eugenia@cococha-gnucita:~/Documentos/documentacion/lisp$ cd ../../../
eugenia@cococha-gnucita:~$

# Atajo para acceder a tu home independientemente de dónde te encuentres
eugenia@cococha-gnucita:~/webprojects/demo.europio.org/public/core/cli$ cd
eugenia@cococha-gnucita:~$ pwd
/home/eugenia

# Un atajo para no escribir /home/usuario
eugenia@cococha-gnucita:~/webprojects/demo.europio.org/public/$ cd ~/Documentos/
eugenia@cococha-gnucita:~/Documentos$
```

Listar documentos y directorios, no se resume solo a `ls`. Algunos trucos:

```
# Listado de documentos con información adicional
eugenia@cococha-gnucita:~/test/$ ls -l
total 12
-rw-r--r-- 1 eugenia eugenia   31 Apr  8 15:07 controller.php
drwxr-xr-x 2 eugenia eugenia 4096 Apr  8 15:07 core
-rw-r--r-- 1 eugenia eugenia    0 Apr  8 15:07 settings.php.dist
drwxr-xr-x 6 eugenia eugenia 4096 Apr  8 15:07 static

# Mostrar también ocultos
eugenia@cococha-gnucita:~/test/$ ls -la
total 24
drwxr-xr-x 4 eugenia eugenia 4096 Apr  8 15:07 .
drwxr-xr-x 4 eugenia eugenia 4096 Apr  8 15:07 ..
-rw-r--r-- 1 eugenia eugenia   31 Apr  8 15:07 controller.php
drwxr-xr-x 2 eugenia eugenia 4096 Apr  8 15:07 core
-rw-r--r-- 1 eugenia eugenia   55 Apr  8 15:07 .htaccess
-rw-r--r-- 1 eugenia eugenia    0 Apr  8 15:07 settings.php.dist
drwxr-xr-x 6 eugenia eugenia 4096 Apr  8 15:07 static

# Tamaño de archivo más legible
eugenia@cococha-gnucita:~/test/$ ls -lah
total 24K
drwxr-xr-x 4 eugenia eugenia 4.0K Apr  8 15:07 .
drwxr-xr-x 4 eugenia eugenia 4.0K Apr  8 15:07 ..
-rw-r--r-- 1 eugenia eugenia   31 Apr  8 15:07 controller.php
drwxr-xr-x 2 eugenia eugenia 4.0K Apr  8 15:07 core
-rw-r--r-- 1 eugenia eugenia   55 Apr  8 15:07 .htaccess
-rw-r--r-- 1 eugenia eugenia    0 Apr  8 15:07 settings.php.dist
drwxr-xr-x 6 eugenia eugenia 4.0K Apr  8 15:07 static

# Listar recursivamente
eugenia@cococha-gnucita:~/test/$ ls -lahR
.:
total 24K
```

```
drwxr-xr-x 4 eugenia eugenia 4.0K Apr  8 15:07 .
drwxr-xr-x 4 eugenia eugenia 4.0K Apr  8 15:07 ..
-rw-r--r-- 1 eugenia eugenia   31 Apr  8 15:07 controller.php
drwxr-xr-x 2 eugenia eugenia 4.0K Apr  8 15:07 core
-rw-r--r-- 1 eugenia eugenia   55 Apr  8 15:07 .htaccess
-rw-r--r-- 1 eugenia eugenia    0 Apr  8 15:07 settings.php.dist
drwxr-xr-x 6 eugenia eugenia 4.0K Apr  8 15:07 static

./core:
total 12K
drwxr-xr-x 2 eugenia eugenia 4.0K Apr  8 15:07 .
drwxr-xr-x 4 eugenia eugenia 4.0K Apr  8 15:07 ..
-rw-r--r-- 1 eugenia eugenia  709 Apr  8 15:07 helpers.php

./static:
total 24K
drwxr-xr-x 6 eugenia eugenia 4.0K Apr  8 15:07 .
drwxr-xr-x 4 eugenia eugenia 4.0K Apr  8 15:07 ..
drwxr-xr-x 2 eugenia eugenia 4.0K Apr  8 15:07 css
drwxr-xr-x 2 eugenia eugenia 4.0K Apr  8 15:07 html
drwxr-xr-x 2 eugenia eugenia 4.0K Apr  8 15:07 img
drwxr-xr-x 2 eugenia eugenia 4.0K Apr  8 15:07 js
...
...
```

Si además se desea ordenar el listado, puede sumarse:

```
-X          Ordenar por extensión
-S          Ordenar por tamaño
-T          Ordenar por fecha y hora
```

También puede obtenerse un listado tipo árbol, instalando `tree`:

```
eugenia@cococha-gnucita:~/test/$ tree
.
├── controller.php
├── core
│   └── helpers.php
├── settings.php.dist
└── static
    ├── css
    ├── html
    │   └── template.html
    ├── img
    └── js
```

Algunas opciones muy útiles de `tree` pueden ser las siguientes:

```
-a          Incluye ocultos
-p          Incluye permisos
-h          Incluye el tamaño de los archivos de forma legible
-u          Incluye el usuario propietario
-g          Incluye el grupo propietario
```

```
-L NUMERO  Para indicar la cantidad máxima de niveles dependencia a mostrar


eugenia@cococha-gnucita:~/test/$ tree -aphug
.
├── [-rw-r--r-- eugenia  eugenia    31]  controller.php
├── [drwxr-xr-x eugenia  eugenia  4.0K]  core
│   └── [-rw-r--r-- eugenia  eugenia   709]  helpers.php
├── [-rw-r--r-- eugenia  eugenia    55]  .htaccess
├── [-rw-r--r-- eugenia  eugenia     0]  settings.php.dist
└── [drwxr-xr-x eugenia  eugenia  4.0K]  static
    ├── [drwxr-xr-x eugenia  eugenia  4.0K]  css
    ├── [drwxr-xr-x eugenia  eugenia  4.0K]  html
    │   └── [-rw-r--r-- eugenia  eugenia  1.4K]  template.html
    ├── [drwxr-xr-x eugenia  eugenia  4.0K]  img
    └── [drwxr-xr-x eugenia  eugenia  4.0K]  js
```

Búsqueda y localización de archivos y programas

Para **buscar archivos** te puedes ayudar del comando `find` el cual posee la siguiente sintaxis:

```
find DONDE OPCIONES
```

`DONDE`, debe ser reemplazado por la ruta de base en la cuál se realizará la búsqueda. Por ejemplo, para buscar archivos en toda mi `home` (incluyendo los subdirectorios), escribo:

```
find /home/eugenia OPCIONES
```

Entre las opciones de búsqueda, la más frecuente es por nombre de archivo. Se puede buscar por el nombre completo:

```
eugenia@cococha-gnucita:~$ find /home/eugenia -name 'PHP Básico.odt'
/home/eugenia/Cursos/editables/PHP Básico.odt
eugenia@cococha-gnucita:~$
```

O también, realizar búsquedas por nombre parcial, utilizando comodines y expresiones regulares:

```
Buscará todos los archivos con extensión .odt
eugenia@cococha-gnucita:~$ find /home/eugenia -name '*.odt'

Buscará todos los archivos que comiencen por la palabra mvc (en minúsculas)
eugenia@cococha-gnucita:~$ find /home/eugenia -name 'mvc*'

Buscará todos los archivos que comiencen por la palabra mvc pero esta vez
```

```
sin distinguir mayúsculas y minúsculas
eugenia@cococha-gnucita:~$ find /home/eugenia -iname 'mvc*'
```

Localizar la ubicación del ejecutable de un programa mediante el comando `which`:

```
eugenia@cococha-gnucita:~$ which apache2
/usr/sbin/apache2
eugenia@cococha-gnucita:~$ which php5
/usr/bin/php5
```

Si además se desea localizar el binario más los archivos fuentes y el manual, se utiliza el comando `whereis`:

```
eugenia@cococha-gnucita:~$ whereis apache2
apache2: /usr/sbin/apache2 /etc/apache2 /usr/lib/apache2 /usr/share/apache2
/usr/share/man/man8/apache2.8.gz
eugenia@cococha-gnucita:~$ whereis python
python: /usr/bin/python2.7 /usr/bin/python /etc/python2.7 /etc/python
/usr/lib/python2.7 /usr/bin/X11/python2.7 /usr/bin/X11/python
/usr/local/lib/python2.7 /usr/include/python2.7_d /usr/include/python2.7
/usr/share/python /usr/share/man/man1/python.1.gz
```

Manipulando archivos y directorios

Para **crear directorios** se utiliza el comando `mkdir`:

```
# Crear un único directorio
eugenia@cococha-gnucita:~/borrador$ mkdir un_solo_directorio

# Crear varios directorios anidados
eugenia@cococha-gnucita:~/borrador$ mkdir -p varios/directorios/anidados
eugenia@cococha-gnucita:~/borrador$ tree varios
varios
└── directorios
    └── anidados

2 directories, 0 files

# Crear más de un directorio al mismo tiempo (no anidados)
eugenia@cococha-gnucita:~/borrador/varios/directorios/anidados$ mkdir dir1 dir2
eugenia@cococha-gnucita:~/borrador/varios/directorios/anidados$ tree
.
├── dir1
└── dir2

2 directories, 0 files
```

Para **crear archivos vacíos** se puede utilizar el comando `touch` como efecto colateral ya que dicho comando está destinado a cambiar la fecha/hora de un archivo, pero si este no existe, lo crea:

```
eugenia@cococha-gnucita:~/borrador/varios$ touch __init__.py
eugenia@cococha-gnucita:~/borrador/varios$ find ~/borrador/varios -name '__init__.py'
/home/eugenia/borrador/varios/__init__.py
```

Como alternativa, puede utilizarse echo, una instrucción *bash* destinada a escribir en pantalla. Dicha instrucción utilizada de forma conjunta con el carácter de direccionamiento de la salida estándar > ayuda indirectamente, a crear un archivo:

```
Crea un archivo vacío
eugenia@cococha-gnucita:~/borrador/varios$ echo '' > archivo.php

Crea un archivo con las etiquetas <?php ?>
eugenia@cococha-gnucita:~/borrador/varios$ echo '<?php ?>' > archivo.php
```

Para **eliminar archivos y directorios**:

```
# Eliminar archivos
eugenia@cococha-gnucita:~/borrador/varios$ rm __init__.py

# Eliminar directorios
eugenia@cococha-gnucita:~/borrador/varios$ rm -R directorios
```

Tener en cuenta que la opción -R siempre indica recursividad de acción en la gran mayoría de los comandos.

Es posible además, **eliminar archivos o directorios de forma segura haciéndolos irrecuperables**. Esto es muy útil cuando por ejemplo hemos guardado información vital (como contraseñas) en un archivo de texto plano y no queremos que el archivo se pueda recuperar mediante un software de recuperación de datos. Este comando sobre-escribirá el archivo tantas veces como le indiquemos y finalmente lo eliminará (es decir, elimina el archivo después de haber sido sobre-escrito N cantidad de veces):

```
eugenia@cococha-gnucita:~/borrador/uno$ shred -n 12 archivo1.copia -u

-n CANTIDAD      Indica la cantidad de veces que el archivo será sobre-escrito
-u               Elimina el archivo tras sobre-escribirlo
```

Para mover o **renombrar archivos y directorios** se utiliza la sintaxis:

```
mv NOMBRE_ACTUAL NUEVO_NOMBRE
```

Por ejemplo:

```
eugenia@cococha-gnucita:~/borrador$ mv varios nuevo_nombre
eugenia@cococha-gnucita:~/borrador$ find ~/borrador/ -name 'nuevo_nombre'
/home/eugenia/borrador/nuevo_nombre
eugenia@cococha-gnucita:~/borrador$ find ~/borrador/ -name 'varios'
eugenia@cococha-gnucita:~/borrador$
```

Es posible mover los archivos o directorios a una ruta distinta con el mismo nombre:

```
eugenia@cococha-gnucita:~/borrador$ mkdir -p uno/dos/tres/cuatro
eugenia@cococha-gnucita:~/borrador$ cd uno/dos/tres
eugenia@cococha-gnucita:~/borrador/uno/dos/tres/$ mv cuatro ../
eugenia@cococha-gnucita:~/borrador/uno/dos/tres/$ cd ..
eugenia@cococha-gnucita:~/borrador/uno/dos/$ tree
.
├── cuatro
└── tres
```

O moverlos a una ruta distinta con distinto nombre:

```
eugenia@cococha-gnucita:~/borrador/uno$ touch archivo1 archivo2
eugenia@cococha-gnucita:~/borrador/uno$ mv archivo2 dos/archivo_dos
eugenia@cococha-gnucita:~/borrador/uno$ tree dos/
dos/
├── cuatro
├── tres
└── archivo_dos
```

TIP: para repetir un comando que ya hayas ejecutado, puedes utilizar varios métodos: 1) flecha arriba y abajo para ir desde el último comando ejecutado hacia atrás y regresar; o 2) mi preferido: pulsando **ctrl + R y escribes los primeros caracteres de un comando ya ejecutado para realizar una búsqueda**. Si tienes que cancelar dicha búsqueda, pulsas Ctrl + G.

Para **copiar archivos** se utiliza la sintaxis:

```
cp ARCHIVO_ORIGINAL NOMBRE_COPIA
```

Por ejemplo:

```
eugenia@cococha-gnucita:~/borrador/uno$ cp archivo1 archivo1.copia
```

Mientras que para **copiar directorios** es necesario hacerlo de forma recursiva con la opción `-R`:

```
cp -R DIRECTORIO_ORIGINAL NOMBRE_COPIA
```

Por ejemplo:

```
eugenia@cococha-gnucita:~/borrador$ cp -R uno/ uno.copia
eugenia@cococha-gnucita:~/borrador$ cd uno.copia/
eugenia@cococha-gnucita:~/borrador/uno.copia/$
```

Trabajando con archivos externos

Muchas veces te será necesario **descargar un archivo desde alguna dirección de Internet**. Pero ¿cómo hacerlo sin un navegador? Es muy simple. Puedes utilizar el comando `wget`:

```
eugenia@cococha-gnucita:~/borrador$ wget
http://www.europio.org/downloads/EuropioEngine_3_0_beta_1--app_demo.tar.gz
--2013-04-08 18:12:26--
http://www.europio.org/downloads/EuropioEngine_3_0_beta_1--app_demo.tar.gz
Resolviendo www.europio.org (www.europio.org)... 66.228.52.93
Conectando con www.europio.org (www.europio.org)[66.228.52.93]:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 109782 (107K) [application/x-gzip]
Grabando a: “EuropioEngine_3_0_beta_1--app_demo.tar.gz”

100%
[==========================================================================================
==========================>] 109.782      146K/s   en 0,7s

2013-04-08 18:12:27 (146 KB/s) - “EuropioEngine_3_0_beta_1--app_demo.tar.gz”
guardado [109782/109782]

eugenia@cococha-gnucita:~/borrador$
```

También es posible **descargar varios archivos de forma recursiva** agregando el argumento `-r`:

```
wget -r http://www.europio.org/
```

Copiar archivos mediante SSH suele ser sumamente útil cuando deseas enviar tus archivos de configuración al servidor sin tener que estar sincronizando carpetas. Para copiar archivos desde tu ordenador al servidor debes utilizar:

```
scp /ruta/de/archivo/local usuario@ip-host-server:/ruta/a/directorio/remoto
```

Por ejemplo:

```
eugenia@cococha-gnucita:~$ scp reporte_apache.html user@123.456.78.90:/home/user
reporte_apache.html                                   100%   34KB  34.0KB/s   00:00
eugenia@cococha-gnucita:~$
```

Y, para "descargar" archivos desde tu servidor hacia tu ordenador local, utilizas la sintaxis inversa:

```
scp user@123.456.78.90:/home/user/archivo.py /home/eugenia/
```

Empaquetado y desempaquetado

En caso que necesites **empaquetar y comprimir** todos los archivos de un directorio, puedes utilizar:

```
tar -czv DIRECTORIO > ARCHIVO.tar.gz
```

Por ejemplo:

```
eugenia@cococha-gnucita:~$ tar -czv borrador/uno/ > directorio-uno.tar.gz
borrador/uno/
borrador/uno/archivo1
borrador/uno/dos/
borrador/uno/dos/tres/
borrador/uno/dos/cuatro/
borrador/uno/dos/archivo_dos
borrador/uno/a
borrador/uno/hola
```

Luego, para **explorar un archivo sin descomprimirlo**, se puede utilizar:

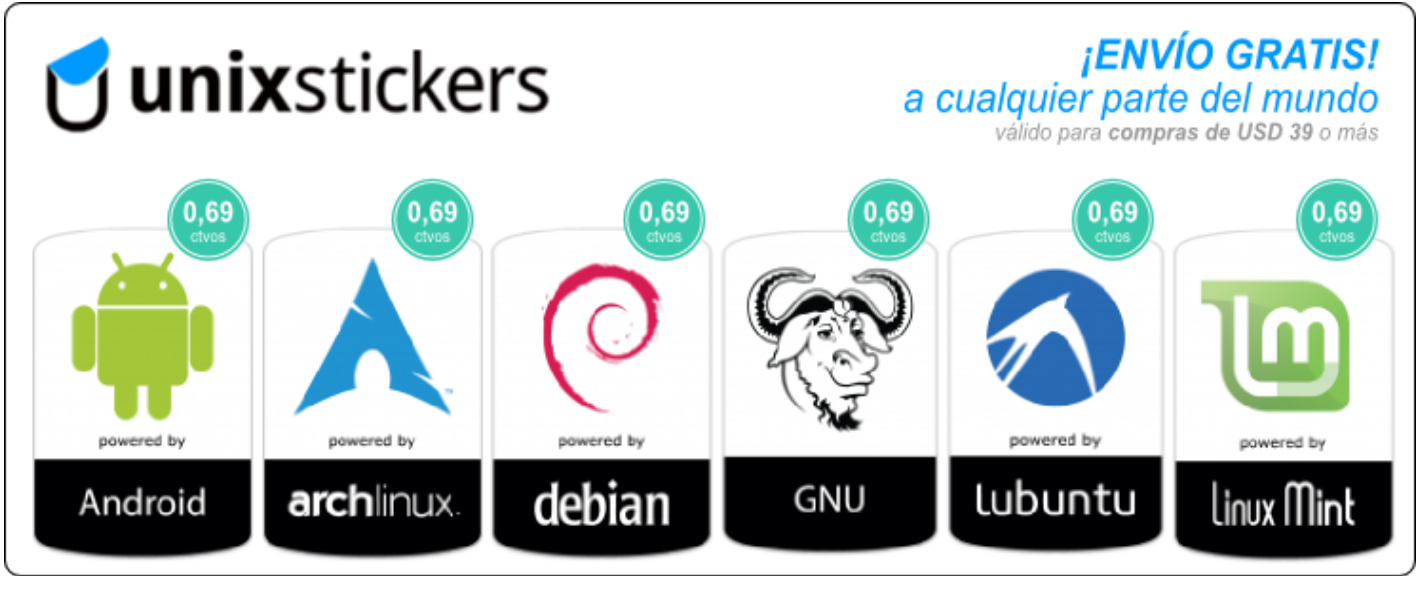
```
eugenia@cococha-gnucita:~$ tar -tf directorio-uno.tar.gz
borrador/uno/
borrador/uno/archivo1
borrador/uno/dos/
borrador/uno/dos/tres/
borrador/uno/dos/cuatro/
borrador/uno/dos/archivo_dos
borrador/uno/a
borrador/uno/hola
```

Finalmente, se podrá **descomprimir un archivo** mediante:

```
eugenia@cococha-gnucita:~$ tar -xzv < directorio-uno.tar.gz
```

Resumen de comandos

Moverse por el sistema de Archivos	
Conocer el directorio actual	`pwd`
Auto-completar el nombre de archivos y directorios	`(pulsar la tecla TAB)`
Moverse hacia adelante	`cd carpeta1/carpeta2`
Moverse hacia atrás	`cd ..`
Volver a la `home`	`cd`
Abreviar `/home/usuario` en una ruta	`~/`
Listar archivos y directorios (método más óptimo)	`ls -lha`
Mostrar árbol de archivos y directorios (requiere instalar el paquete `tree`)	`tree -a`
Búsqueda y localización de archivos y programas	
Buscar archivos	`find DONDE -iname 'PATRON'`
Localizar un programa	`which PROGRAMA`
Ubicar un programa, su binario, fuentes y manual	`whereis PROGRAMA`
Manipulación de archivos y directorios	
Crear archivo	`touch ARCHIVO`
Crear directorios	`mkdir -p DIRECTORIO_1/ DIRECTORIO_2`
Eliminar archivos	`rm ARCHIVO`
Eliminar directorio	`rm -R DIRECTORIO`
Eliminación segura de archivos	`shred ARCHIVO -n CANTIDAD -u`
Renombrar archivos o directorios	`mv ORIGINAL NUEVO_NOMBRE`
Copiar archivo	`cp ORIGINAL NUEVO_NOMBRE`
Copiar directorio	`cp -R ORIGINAL NUEVO_NOMBRE`
Manejo de archivos externos	
Descargar archivo de Internet	`wget URL`
Descargar Web de forma recursiva	`wget -r URL`
Copiar archivos de local a servidor (el comando se ejecuta desde la PC local)	`scp /ruta/local user@ip:/ruta/destino`
Copiar archivos desde el servidor a la PC local (el comando se ejecuta desde la PC local)	`scp user@ip:/ruta/server /ruta/local`
Empaquetado y Desempaquetado	
Empaquetar	`tar -czv DIRECTORIO > tarball.tar.gz`
Explorar un *tarball*	`tar -tf tarball.tar.gz`
Desempaquetar en el directorio actual	`tar -xzv < tarball.tar.gz`
Misceláneas: historial de comandos ejecutados en la terminal	
Moverse en el historial de comandos ejecutados	`(flecha arriba / flecha abajo)`
Buscar un comando ejecutado, en el historial	`(Teclas Ctrl + R)`
Cancelar la búsqueda iniciada	`(Teclas Ctrl + G)`



¿El enlace de la imagen no funciona? Copia y pega esta URL: http://www.unixstickers.com/?tracking=519a6cd14bcdb

¿El enlace de la imagen no funciona? Copia y pega esta URL: http://www.hdmagazine.org/donar

El mundo de la criptografía y la psicología de las contraseñas

La criptografía tiene principios tan interesantes que aveces son pasados por alto, pero que sin embargo el aprender sobre ellos nos dan la explicación de muchas situaciones cotidianas. Aunque no parezca, existe toda una explicación psicológica detrás del hecho de elegir una contraseña. Proteger nuestra información debe ser una responsabilidad y un derecho para todos y por ello existe una herramienta libre llamada KeePass: un gestor que nos aliviará el lidiar con tantas claves.

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

Estudiante de Ingeniería Informática. Miembro 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**

Aunque no lo parezca, la criptografía está presente en nuestra vida cotidiana, ya que a diario usamos claves para producir la salida funcional de algoritmos criptográficos. En palabras simples, usamos claves para verificar si estamos o no autorizados para acceder a algún servicio o sistema. Aunque este parezca un tema sencillo existe toda una psicología detrás, como también criterios que deben tomarse en cuenta para proteger nuestra información y más detalles que muchas veces pasamos por alto pero que en cambio si nos tomáramos el tiempo se registrarían menos casos de robo y/o infiltración de información.

El principio de Kerckhoff

Que nuestra información se mantenga protegida debería ser para todos un derecho, sin embargo existen personas movidas por intereses de por medio que buscarán la manera de obtener estos datos. Uno de los puntos importantes a tomar en cuenta es lo que el Principio de Kerckhoff dice: "sólo el mantener la clave en secreto proporciona seguridad" ya que como Claude Shannon dijo: "El enemigo conoce el sistema"; debemos estar prevenidos para todo tipo de ataque y no llegar a ser sorprendidos, al contrario, adelantarnos a lo que pueda pasar y de esta manera lograr que nuestros mecanismos sean mejores y mantengan nuestra información a salvo. Mantener las claves en secreto es uno de los problemas más difíciles. Esta pequeña pieza puede proteger todo un enorme sistema y por esto se considera como uno de los eslabones más débiles para un supuesto atacante, ya que si la obtiene accede inmediatamente a todos los datos cifrados y se trae abajo todo lo que protegía; es cierto también que por este motivo cada vez se toman más medidas y precauciones para proteger grandes sistemas, quizás ya no solo una o dos claves, sino otros conceptos como la verificación en dos o tres pasos, la biometría y más.

Este es un texto cifrado: LEGOIVW ERH HIZIPSTIVW. Este criptosistema tiene como clave sumar 4 letras al texto normal, por lo que resuelto, la frase descifrada sería: HACKERS AND DEVELOPERS.

El secreto perfecto

La longitud de la clave es el tamaño medido en bits, ya que la criptografía moderna maneja medidas binarias; este aspecto es determinante ya que mientras más pequeña sea, la susceptibilidad de un cifrador será mayor frente a un ataque de búsqueda exhaustivo. Una clave debería ser tan grande como para que un ataque de fuerza bruta sea imposible, este ataque se refiere a obtener una clave probando todas las combinaciones posibles hasta encontrar la correcta, la longitud debería ocasionar que este ataque lleve demasiado tiempo al ejecutarlo y finalmente no obtener el dato que buscaba, esto es considerado el secreto perfecto; para explicar esto con un ejemplo sencillo, una clave de longitud de n bits tiene "2 a la n" claves posibles, por eso es que mientras más extensa sea más alto será el grado de dificultad de poder obtenerla.

> *"Las organizaciones gastan millones de dólares en firewalls y dispositivos de seguridad, pero tiran el dinero porque ninguna de estas medidas cubre el eslabón más débil de la cadena de seguridad: la gente que usa y administra los ordenadores" – Kevin Mitnick*

Psicología de las contraseñas

Existe una interesante intersección entre criptografía y psicología, lo que da como resultado la psicología de la contraseña, que estudia qué es lo que hace a las claves fáciles de recordar o adivinar, ya que para que una contraseña funcione exitosamente es importante que el usuario la memorice. La psicología detrás de escoger una clave es un balance único entre memorización, seguridad y conveniencia; las contraseñas pueden ser reflejo de la personalidad: aquellos que sean más orientados a la seguridad pueden elegir las más complicadas mientras que los que se sienten muy seguros con su vida cotidiana pueden nunca cambiarla. La memorización está relacionada con el uso de la mnemotecnia pero para algunas personas, esto se vuelve un tema complicado y la solución a ello es tener ciertas políticas que permitan mantener las contraseñas seguras.

Políticas de seguridad

Existen muchos factores en la seguridad que debemos tener en cuenta al hablar de contraseñas. Una política es un conjunto de reglas para mejorar la seguridad haciendo que los usuarios empleen claves seguras.

Una de las políticas es acerca de la **longitud y formación**. Se recomienda:

- usar letras mayúsculas y minúsculas;
- incluir números y caracteres especiales;
- evitar el uso de palabras que se encuentren en el diccionario o incluir información personal;
- evitar el uso de números de teléfono u otros números comunes;

Otra política es respecto a la **caducidad** de la contraseña. Se recomienda que se proponga un determinado tiempo para cambiarla que puede ser de 90 o 120 días. Así, se podrían prevenir muchos ataques que requieren de tiempo para obtener esta pieza de información.

La siguiente política nos habla sobre la **gestión de las claves**. Se recomienda:

- nunca usar la misma contraseña para más de una cuenta;
- nunca proporcionar la contraseña a nadie;
- nunca escribirla en un papel y dejarla cerca de la computadora;
- ser cuidadoso de desconectarse del servicio del que estábamos haciendo uso al finalizar; entre muchas más.

Actualmente existen muchas formas de atentar contra la seguridad de la información. Uno de los delitos más conocidos es el *phishing*. Éste está dentro de las estafas cibernéticas ya que es cometido mediante el uso de ingeniería social. Al querer adquirir información de forma fraudulenta, el *phisher* (el estafador) se hace pasar por una persona de confianza de alguna empresa y por correo electrónico o por llamadas telefónicas intenta obtener los datos que desea.

Ingeniería Social

Es importante entender como funciona la ingeniería social. Ésta es la práctica de obtener información confidencial manipulando a los usuarios; el principio que la sustenta es que en todo sistema el eslabón débil es el usuario y los daños causados varían entre la pérdida del acceso al correo electrónico y pérdidas económicas sustanciales. Por ejemplo, la técnica con la cual debemos tener más cuidado es cuando llega un correo electrónico en el cual dan una ruta manipulada para que parezca la original de algún servicio que usemos. Quizás es un e-mail del banco al que se está afiliado, pero en realidad es un enlace que nos estará dirigiendo a solicitarnos el usuario y la contraseña de nuestra cuenta y según muchos estudios las personas suelen ser muy confiadas de indicar sus datos sin pensar que esto se trata de una estafa; otro caso puede ser al recibir una llamada telefónica de parte de alguna persona de confianza de la empresa u organización en la que trabaja quien en realidad es un estafador que averiguó cierta información y ahora quiere utilizar eso para obtener los datos que desea.

Cracking de cuentas y robo de información

El *cracking* de contraseñas puede definirse como la acción de corromper y destruir las medidas de seguridad que protegen la privacidad de una contraseña. Muchas personas no se dan el tiempo de buscar cómo crear una contraseña segura, es más, el problema radica en que pueden usar la misma clave para todas sus cuentas, pero deben pensar que si un *Cracker* llega a obtener esa información tendrá acceso a todo y esto, es sumamente peligroso ya que puede hacerse pasar por uno, enviar correos electrónicos desprestigiando su centro laboral u ocasionándole conflictos de otra índole. Pero por otro lado, también es cierto que el motivo principal es que si se coloca una clave distinta para cada servicio es muy probable que se las pueda olvidar.

Contraseñas fuertes para cada servicio

Para darle solución a esto existen muchas maneras que podemos empezar a aplicar para poner a salvo y asegurar nuestra información. Un método eficaz para crear una contraseña segura es elegir una frase (de una canción, del trabajo, etc), luego tomar la primera letra de cada palabra y finalmente cambiar alguna de ellas por números o caracteres especiales; de esta manera la contraseña es larga y será mucho más complicada de ser adivinada.

La frase: Hackers & Developers - Magazine digital de distribución mensual sobre Software Libre, Hacking y Programación. Resultaría en: H&D-mdddmsslhp. Y añadiendo ciertos caracteres obtendríamos lo siguiente: H&D-mdDdm5SLHP.

Otro buen método es que ya teniendo una base se creen contraseñas distintas para cada servicio que usemos, por ejemplo, según las letras obtenidas anteriormente añadirle algo que nos indique de que servicio se trata.

Si su contraseña predeterminada es: H&D-mdDdm5SLHP. Por ejemplo, se puede crear una contraseña para gmail: H&D-mdDdm5SLHP-gmail o una contraseña para twitter: H&D-mdDdm5SLHP-twitter

Otra técnica de crear contraseñas podría ser colocando una base intermedia.

Si por ejemplo usamos una frase personal para describirnos: Free Software activist and Python developer. Podríamos tener como clave fsaapd, agregando caracteres especiales obtendríamos f544pd y para usar una para cada servicio podriamos usar la base intermedia, por ejemplo para gmail obtendríamos: GMAf544pdIL o si fuera para twitter TWIf544pdTTER.

Otra técnica para la creación de claves puede ser la base final.

Si tenemos como un día importante el 27 de Mayo del 2013 (Lanzamiento de hdmagazine n° 7) podríamos tener como clave 270513, pero para que no sea algo común podríamos cambiar algunos números por letras por ejemplo: 27os1e y de esta manera para las personas que les agrada tanto usar fechas la pueden mantener más segura y luego para usar una para cada servicio podriamos usar la base final, por ejemplo para gmail obtendríamos: GmaiL27os1e o si fuera para twitter TwitteR27os1e.

"Saber romper medidas de seguridad no hace que seas hacker, al igual que saber hacer un puente en un coche no te convierte en un ingeniero de automoción". - Eric Raymond

KeePass Password Safe

KeePass[1] es un gestor de contraseñas fácil de usar, liviano, libre y de código abierto. La

1 http://keepass.info

lista de contraseñas que las personas tienen que usar es interminable y este gestor ayuda a administrar las claves de manera segura, pudiendo colocarlas todas en una base de datos, protegiéndolas con una llave maestra (de esta manera solo habría que recordar la clave maestra para desbloquear la base de datos entera).

Estas bases de datos están encriptadas (no solo los campos de contraseñas, sino también nlos ombres de usuario, notas, etc) usando los mejores y más seguros algoritmos actualmente conocidos: AES y Twofish. También se pueden usar archivos de clave que son más seguros que la clave maestra, pero tendremos que tener mucho cuidado de no perder el dispositivo donde se tiene almacenado el archivo; para contar con mayor seguridad se pueden combinar ambos métodos: la contraseña y el archivo de clave. KeePass es portable, no almacena nada en la computadora, ni claves de registro ni archivos de inicialización. La lista de contraseñas se puede exportar en diferentes formatos como TXT, HTML, XML y CSV.

KeePass puede ser una gran solución al problema de la gestión de contraseñas. Esta herramienta brinda muchas facilidades a los usuarios y sobretodo se puede considerar como otra política que podemos ponernos como guía para protegernos contra el robo y/o infiltración a nuestra información.

Las personas muchas veces se quejan del robo de su información o de la pérdida de sus cuentas, sin darse cuenta que ellos mismos son quienes ocasionan este tipo de situaciones y malas experiencias. Por más que uno piense que la información que tiene en determinado servicio no es muy importante y que no hay problema si llegan a acceder a ella es mejor protegerla de todas maneras o si no vas a usar esa cuenta, eliminarla.

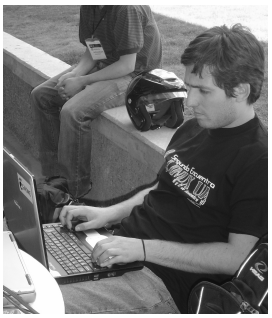
Es importante contar con una contraseña segura y distinta para cada servicio y de una u otra forma seguir todas las indicaciones redactadas líneas arriba, aplicar nuestras propias políticas de seguridad y cumplirlas ya que la mejor defensa contra la ingeniería social es educar y entrenar a los usuarios y sobretodo que ninguno sienta que nunca va ser víctima de un *cracker* porque sí podría pasar.

Evitar malos ratos e inconvenientes protegiendo nuestra información de esta simple manera es algo a lo que definitivamente debemos darle la importancia que merece.

Conociendo a Vala

Vala es un "nuevo" lenguaje de programación multiparadigma[2], cuya sintaxis es bastante similar a C#, pero modificado para ser utilizado por GObject.

Escrito por: **Fabio Durán Verdugo –** Colaborador invitado



Fabio es **Ingeniero de Software** y miembro de GNOME Foundation y GNOME Chile. Colabora en el BugSquad team de GNOME. Amante y propulsor del Software Libre y código abierto. Actualmente, también está dedicado a la docencia universitaria. *Python lover*.

Perfiles:
https://live.gnome.org/FabioDuran

Twitter: @fabioduran

Muchos desarrolladores quieren escribir aplicaciones para GNOME -o bibliotecas- en lenguajes de programación de alto nivel, pero no pueden o no quieren utilizar Python o Java, por diversas razones. Entre la más común de las justificaciones, es que requieren de un *wrapper* o *binding* para poder acceder a las bibliotecas.

Ante ello la única forma que nos queda para programar en ambiente GNOME, es usando la API nativa de C, con la limitante de no contar con apoyo de una sintaxis o API para GObject[3].

El fundamento de Vala

Vala fue diseñado y orientado para utilizar las librerías existentes de C, especialmente basadas en GObjects, una adaptación entre lenguajes de programación, ya que por ejemplo no necesita ningún requisito extra en tiempo de ejecución y además de no requerir una ABI (*application binary interface* - interfaz binaria de aplicación[4]) diferente en comparación a las aplicaciones y bibliotecas escritas en C, como ocurría antes de su creación.

A lo anterior podemos agregar el enfoque del desarrollo definido por los diseñadores Vala, que plantea su fácil utilización, debido a que lo único que se necesita es una

2 http://es.wikipedia.org/wiki/Lenguaje_de_programación_multiparadigma
3 https://developer.gnome.org/gobject/
4 http://en.wikipedia.org/wiki/Application_binary_interface

librería tipo Vala, o en otras palabras, una API que contenga clases y métodos en sintaxis Vala.

Instalando

Para instalar Vala en tu máquina lo puedes realizar a través de:

```
apt
$ apt-get install vala

yum
yum install vala

pacman
pacman -S vala
```

El compilador

El compilador de Vala es `valac`. Traduce el código fuente, permitiendo a los desarrolladores escribir código orientado a objetos complejo con rapidez, manteniendo una API y ABI de C estándar con un bajo uso de memoria.

El uso del **compilador**:

```
$ valac <fuente.vala> -o <binario>
```

Para **ejecutar nuestro binario** generado solo sería:

```
$ ./<binario>
```

Nuestro Primer Programa en Vala: el clásico Hola Mundo!

Para escribir código Vala se puede utilizar el editor de texto o IDE que más te acomode, como por ejemplo vim.

Archivo: `holamundo.vala`

```
class HolaMundo : GLib.Object {

    public static int main(string[] args) {

        stdout.printf("Hola Mundo!\n");
```

```
        return 0;
    }
}
```

Compilamos:

```
valac holamundo.vala -o holamundo
```

Ejecutamos:

```
$ ./holamundo
```

Y la salida será:

```
Hola Mundo!
```

Ya tenemos nuestro primer hola mundo y podemos ahora presumir que tenemos nuestro primer programa Vala ¿no es fascinante? Pero creo que sería aún mejor si creamos un hola mundo con una interfaz gráfica.

Nuestro Primer Programa Vala usando la librería gráfica de GNOME (GTK+)

Archivo: `holamundo_gtk.vala`

```
using Gtk;

int main (string[] args) {
    Gtk.init (ref args);

    var window = new Window ();
    window.title = "Nuestro Primer Programa Vala GNOME";
    window.window_position = WindowPosition.CENTER;
    window.set_default_size (400, 100);

    var button = new Button.with_label ("Haz click y di Hola Mundo!");
    button.clicked.connect (() => {
    stdout.printf("Hola Mundo!\n");
    });

    window.destroy.connect (Gtk.main_quit);
    window.add (button);
    window.show_all ();
```

```
    Gtk.main ();
    return 0;
}
```

Compilamos:

```
$ valac --pkg gtk+-3.0 holamundo_gtk.vala -o holamundo_gtk
```

La instrucción `--pkg gtk+-3.0`, básicamente le está diciendo al compilador (`valac`) que enlace la librería `gtk+-3.0` a nuestro nuevo archivo binario llamado `holamundo_gtk`.

Usando el compilado correcto podemos ahora ejecutar:

```
$ ./holamundo_gtk
```

La salida al pulsar el ratón, será:

```
Hola Mundo!
```

Por contraparte sin la instrucción `--pkg`, el compilador no reconocerá la librería GTK que estamos instanciando con la instrucción `using Gtk` en nuestro código.

Si lo compiláramos sin este *flag*:

```
$ valac holamundo_gtk.vala -o holamundo_gtk
```

Obtendríamos el siguiente mensaje de error:

```
holamundo_gtk.vala:1.7-1.9: error: The namespace name `Gtk' could not be found
using Gtk;
```

Pues ya está. Tenemos nuestra primera APP Vala Gtk+.

Refactoring: avanzando con las soluciones

En la edición anterior de Hackers & Developers Magazine hicimos una introducción a la técnica de *Refactoring* y planteamos soluciones a problemas cotidianos. En esta entrega final, veremos problemas más complejos que también pueden ser solucionados implementando la técnica de *Refactoring*.

Escrito por: **Eugenia Bahit** (GLAMP Hacker & eXtreme Programmer)

Eugenia es **Arquitecta de Software**, docente e 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**, **The Linux Foundation** y **Debian Hackers**. Creadora de python-printr, Europio Engine y colaboradora de Vim.

Webs:
Cursos de programación: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

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

En la edición anterior, comentábamos la diferencia entre problema y error, explicando que la técnica de *Refactoring* había sido pensada para solucionar problemas pero no, para corregir errores.

Los problemas están muy ligados a la legibilidad del código y su reutilización. Por lo tanto, el *Refactoring* tiene por objetivo, hacer el código más legible y reutilizable.

En esta entrega nos enfocaremos en problemas de legibilidad y reutilización, algo más complejos que los que vimos en el capítulo anterior.

PROBLEMA: Métodos extensos

No solo una expresión puede ser extensa. Muchas veces, nos encontraremos con métodos con extensos algoritmos que realizan varias acciones:

```
function suscribir_al_newsletter() {
    if(isset($_POST['email'])) {
        $email = $_POST['email'];
    } else {
        $email = '';
    }

    if(isset($_POST['nombre'])) {
        $nombre = $_POST['nombre'];
    } else {
        $nombre = '';
    }

    if($email == '') {
        $errors[] = 'E-mail no puede estar vacío';
    elseif(strlen($email) < 7) {
        $errors[] = 'El e-mail no puede tener menos de 7 caracteres';
    } else {
        $email = strtolower(strip_tags($email));
    }

    if(strlen($nombre) < 5) {
        $errors[] = 'Por favor, indique un nombre y apellido legítimo';
    } else {
        $nombre = strtoupper(strip_tags($nombre));
    }

    $id = 0;
    $msg = 'Error desconocido';

    if(isset($errors)) {
        $msg = implode('<br/>', $errors);
    } else {
        $sql = 'INSERT INTO newsletter (nombre, email) VALUES (?, ?);
        $data = array('ss', "{$nombre}", "{$email}");
        $id = DBLayer::ejecutar();
    }

    if($id > 0) {
        $msg = 'Se ha suscrito al newsletter';
    }

    $plantilla = file_get_contents('system_msg.html');
    print str_replace('{MSG}', $msg, $plantilla);
}
```

Cuando existen métodos tan extensos, probablemente, la solución consista en la combinación de diversas técnicas, que van desde agrupar expresiones en una misma línea hasta evitar la asignación de variables temporales (como vimos la edición pasada) y extraer código llevándolo a diferentes métodos. En estos casos, donde más de una técnica puede ser necesaria, un buen comienzo será detectar la cantidad de acciones y responsabilidades que el método o función presentan actualmente.

Si observamos el código anterior, podremos encontrar las siguientes acciones y responsabilidades:

1. Recibir dos datos desde un formulario asignando dicho valor a dos variables;
2. Validar que los datos recibidos desde el formulario sean correctos, retornando un mensaje de error en caso contrario;

3. Filtrar los datos recibidos desde el formulario;
4. Guardar los datos en la base de datos;
5. Mostrar un mensaje en pantalla.

No hace falta demasiado para ver que una sola función, está realizando el trabajo de 5 funciones. Es entonces, cuando procedemos a extraer el código y crear nuevos métodos. Para ello, primero pasamos los bloques de códigos (agrupados por responsabilidad) a nuevos métodos:

```
function recibir_datos() {
    if(isset($_POST['email'])) {
        $email = $_POST['email'];
    } else {
        $email = '';
    }

    if(isset($_POST['nombre'])) {
        $nombre = $_POST['nombre'];
    } else {
        $nombre = '';
    }
}

function validar_datos() {
    if($email == '') {
        $errors[] = 'E-mail no puede estar vacío';
    elseif(strlen($email) < 7) {
        $errors[] = 'El e-mail no puede tener menos de 7 caracteres';
    } else {
        $email = strtolower(strip_tags($email));
    }

    if(strlen($nombre) < 5) {
        $errors[] = 'Por favor, indique un nombre y apellido legítimo';
    } else {
        $nombre = strtoupper(strip_tags($nombre));
    }
}

function guardar_datos() {
    $id = 0;
    $msg = 'Error desconocido';

    if(isset($errors)) {
        $msg = implode('<br/>', $errors);
    } else {
        $sql = 'INSERT INTO newsletter (nombre, email) VALUES (?, ?);
        $data = array('ss', "{$nombre}", "{$email}");
        $id = DBLayer::ejecutar();
    }

    if($id > 0) {
        $msg = 'Se ha suscrito al newsletter';
    }
}

function mostrar_mensaje() {
    $plantilla = file_get_contents('system_msg.html');
    print str_replace('{MSG}', $msg, $plantilla);
}
```

```
function suscribir_al_newsletter() {
    recibir_datos();
    validar_datos();
    guardar_datos();
    mostrar_mensaje();
}
```

Acto seguido, verificamos aquellas variables de uso temporal que hayan quedado definidas pero que sean requeridas por más de una función y las convertimos en propiedades de clase:

```
function recibir_datos() {
    if(isset($_POST['email'])) {
        $this->email = $_POST['email'];
    } else {
        $this->email = '';
    }

    if(isset($_POST['nombre'])) {
        $this->nombre = $_POST['nombre'];
    } else {
        $this->nombre = '';
    }
}

function validar_datos() {
    if($this->email == '') {
        $this->errors[] = 'E-mail no puede estar vacío';
    elseif(strlen($this->email) < 7) {
        $this->errors[] = 'El e-mail no puede tener menos de 7 caracteres';
    } else {
        $this->email = strtolower(strip_tags($this->email));
    }

    if(strlen($this->nombre) < 5) {
        $this->errors[] = 'Por favor, indique un nombre y apellido legítimo';
    } else {
        $this->nombre = strtoupper(strip_tags($this->nombre));
    }
}

function guardar_datos() {
    $id = 0;
    $this->msg = 'Error desconocido';

    if(isset($this->errors)) {
        $this->msg = implode('<br/>', $errors);
    } else {
        $sql = 'INSERT INTO newsletter (nombre, email) VALUES (?, ?);
        $data = array('ss', "{$this->nombre}", "{$this->email}");
        $id = DBLayer::ejecutar();
    }

    if($id > 0) {
        $this->msg = 'Se ha suscrito al newsletter';
    }
}

function mostrar_mensaje() {
```

```
    $plantilla = file_get_contents('system_msg.html');
    print str_replace('{MSG}', $this->msg, $plantilla);
}

function suscribir_al_newsletter() {
    recibir_datos();
    validar_datos();
    guardar_datos();
    mostrar_mensaje();
}
```

Finalmente, aquellas instrucciones que puedan definirse en una sola línea, pueden ser *refactorizadas*:

```
function recibir_datos() {
    $this->email = isset($_POST['email']) ? $_POST['email'] : '';
    $this->nombre = isset($_POST['nombre']) ? $_POST['nombre'] : '';
}

function validar_datos() {
    if($this->email == '') {
        $this->errors[] = 'E-mail no puede estar vacío';
    elseif(strlen($this->email) < 7) {
        $this->errors[] = 'El e-mail no puede tener menos de 7 caracteres';
    } else {
        $this->email = strtolower(strip_tags($this->email));
    }

    if(strlen($this->nombre) < 5) {
        $this->errors[] = 'Por favor, indique un nombre y apellido legítimo';
    } else {
        $this->nombre = strtoupper(strip_tags($this->nombre));
    }
}

function guardar_datos() {
    $id = 0;
    $this->msg = 'Error desconocido';

    if(isset($this->errors)) {
        $this->msg = implode('<br/>', $errors);
    } else {
        $sql = 'INSERT INTO newsletter (nombre, email) VALUES (?, ?);
        $data = array('ss', "{$this->nombre}", "{$this->email}");
        $id = DBLayer::ejecutar();
    }

    if($id > 0) $this->msg = 'Se ha suscrito al newsletter';
}

function mostrar_mensaje() {
    $plantilla = file_get_contents('system_msg.html');
    print str_replace('{MSG}', $this->msg, $plantilla);
}

function suscribir_al_newsletter() {
    recibir_datos();
    validar_datos();
    guardar_datos();
    mostrar_mensaje();
```

```
}
```

PROBLEMA: Código duplicado en una misma clase

Es frecuente -y de lo más común-, que las mismas expresiones, comiencen a duplicarse en diferentes métodos de una misma clase:

```
def set_txt_encabezado(self):
    with open('plantilla.tmpl', 'r') as archivo:
        TEXTO = archivo.read()

    a = TEXTO.replace('X', self.destinatario)
    return "Estimado %s:<br/>" % a

def set_txt_certificacion(self):
    with open('plantilla.tmpl', 'r') as archivo:
        TEXTO = archivo.read()

    a = TEXTO.replace('X', self.destinatario)
    return "Certifica que el %s" % a
```

Las expresiones duplicadas en el código de los diferentes métodos de una misma clase, se solucionan extrayendo el código duplicado de los métodos y colocándolo en un nuevo método de clase:

```
def render(self):
    with open('plantilla.tmpl', 'r') as archivo:
        TEXTO = archivo.read()

    return TEXTO.replace('X', self.destinatario)


def set_txt_encabezado(self):
    return "Estimado %s:<br/>" % self.render()

def set_txt_certificacion(self):
    return "Certifica que el %s" % self.render()
```

PROBLEMA: Código duplicado en clases con la misma herencia

El caso anterior puede darse también, cuando el código se encuentra duplicado en diferentes métodos de clases con la misma herencia:

```
class Carta(Documento):

    #...

    def set_txt_encabezado(self):
        with open('plantilla.tmpl', 'r') as archivo:
```

```
            TEXTO = archivo.read()

        a = TEXTO.replace('X', self.destinatario)
        return "Estimado %s:<br/>" % a


class Certificado(Documento):

    #...

    def set_txt_certificacion(self):
        with open('plantilla.tmpl', 'r') as archivo:
            TEXTO = archivo.read()

        a = TEXTO.replace('X', self.destinatario)
        return "Certifica que el %s" % a
```

En estos casos, en los cuáles existen dos o más clases que heredan de la misma madre, se extrae el código duplicado en los métodos de las clases hijas y con éste, se crea un nuevo método en la clase madre:

```
class Documento(object):

    #...

    def render(self):
        with open('plantilla.tmpl', 'r') as archivo:
            TEXTO = archivo.read()

        return TEXTO.replace('X', self.destinatario)


class Carta(Documento):

    #...

    def set_txt_encabezado(self):
        return "Estimado %s:<br/>" % self.render()


class Certificado(Documento):

    #...

    def set_txt_certificacion(self):
        return "Certifica que el %s" % self.render()
```

PROBLEMA: Código duplicado en varias clases sin la misma herencia

Como era de esperarse, el código también podría aparecer duplicado en diferentes clases pero que no tienen la misma herencia:

```
class Carta {
```

```
    #...

    function set_txt_encabezado() {
        $texto = file_get_contents('plantilla.tmpl');
        $a = str_replace('X', $this->destinatario, $texto);
        return "Estimado $a:<br/>";
    }
}


class Certificado {

    #...

    function set_txt_certificacion() {
        $texto = file_get_contents('plantilla.tmpl');
        $a = str_replace('X', $this->destinatario, $texto);
        return "Certifica que el $a";
    }

}
```

En estos casos, la solución es extraer el código duplicado, crear una nueva clase y con el código extraído, crear un método para esta nueva clase que podrá ser heredada por las anteriores o simplemente, instanciada:

```
class Documento {

    # ...

    function render() {
        $texto = file_get_contents('plantilla.tmpl');
        return str_replace('X', $this->destinatario, $texto);
    }

}

class Carta extends Documento {

    #...

    function set_txt_encabezado() {
        return "Estimado {$this->render()}:<br/>";
    }
}


class Certificado extends Documento {

    #...

    function set_txt_certificacion() {
        return "Certifica que el {$this->render()}";
    }

}
```

¿El enlace de la imagen no funciona? Copia y pega esta URL: http://bit.ly/promo-payoneer

Inception Deck en el Agilismo

Antes de iniciar un proyecto de desarrollo utilizando metodologías ágiles, deben responderse preguntas cruciales que asegurarán un buen comienzo y probablemente el éxito al finalizarlo.

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

Ingeniero Informático con estudios de **Master de Dirección Estratégica en TI**. Senior Software Developer en Belatrix Software Factory**,** activista, contribuidor y consultor de proyectos **FLOSS** y escritor de artículos y libros técnicos de programación.
Perfiles:
http://about.me/neosergio
Twitter: **@neosergio**

Muchos proyectos están destinados a fracasar desde el principio, sobre todo porque no se hacen las preguntas iniciales adecuadas o incluso porque no se tiene el valor para hacerlas.

Para asegurar que exista un buen inicio, existe una herramienta llamada **Inception Deck**, que consiste en 10 preguntas que deben responderse antes de iniciar cualquier proyecto de software.

Herramienta propuesta por Jonathan Rasmusson[5] y que es utilizada ampliamente por diversos equipos ágiles alrededor del mundo.

Diferentes perspectivas

Casi siempre al inicio de un proyecto las personas tienen diferentes perspectivas del proyecto, algunas mas simples otras mas complejas, y no basta asumir con que hay un consenso.

En los proyectos que he participado siempre hemos usado las siguientes estrategias para evitar que esta diversidad de ideas se normalice.

- Explicar a los miembros del equipo el contexto del proyecto.

5 http://agilewarrior.wordpress.com/

- Explicar los objetivos generales y la visión
- Dar a los patrocinadores del proyecto la información exacta que requieren para tomar las decisiones con respecto al proyecto

Estas estrategias, permiten que el equipo de desarrollo tenga claro hacía donde se quiere llegar y apuntar todos los esfuerzos para ello, así como a los patrocinadores permitir aterrizar la idea, teniendo en cuenta las limitaciones actuales tanto tecnológicas como de concepto.

Las preguntas engañosas

En algunos proyectos empiezan a hacerse análisis de factibilidad, de fortalezas y debilidades, de oportunidades y amenazas o incluso estudios estadísticos, pero todos estos análisis y estudios en la mayoría de ocasiones son una perdida de tiempo, porqué dejan de evaluar aspectos críticos del proyecto.

Las siguientes preguntas son las mas frecuentes y engañosas:

- ¿Quienes son los responsables del proyecto?
- ¿Cuanta experiencia tiene el equipo?
- ¿Cuanto dinero tenemos disponible para gastar?
- ¿En que otros proyectos han trabajado?
- ¿Qué lenguaje de programación van a usar y/o herramientas?
- ¿Han hecho algo similar antes?

Todas estas preguntas, brindaran respuestas ambiguas y por lo general darán una perspectiva falsa de control y éxito.

Inception Deck

Consiste en juntar a las personas adecuadas y hacerles las preguntas adecuadas, el sólo hecho de poder intentarlo las primeras veces, asegura un buen inicio, mientras se va repitiendo el mismo procedimiento proyecto tras proyecto, se llega a dominar la técnica y por lo tanto un buen inicio de todo proyecto.

Las preguntas a responder o dudas que resolver en el Deck son:

Preguntar el porque estamos aquí.

Es una manera de recordar quienes somos, cuál es la responsabilidad que tenemos ante

los clientes y porque decidimos formar parte del proyecto.

Crear un elevator pitch

Debemos ser capaces de describir el proyecto en dos oraciones y en menos de un minuto.

Diseñar un eslogan para el producto

Si tendríamos que vender el producto en una revista o banner publicitario, como lo haríamos, saber qué es lo más importante para decir, para asegurar su venta.

Crear una lista de cosas que NO se harán

Una cosa es saber que haremos en el proyecto, pero una parte igual de importante es saber que NO se hará en el proyecto, y ser muy claros en definir los límites.

Conocer a los vecinos.

Porque el proyecto depende de personas, es importante tratar de conocer a los que participarán del proyecto, para poder congeniar y encontrar un ambiente adecuado de trabajo.

Mostrar la solución

Un prototipo o bosquejo de la solución es necesaria para que el equipo conozca la arquitectura de la solución y asegurarse que todos estén visualizando la misma perspectiva.

Preguntar que nos mantienen intranquilos

En el proyecto hay cosas que nos mantienen temerosos, como el desconocimiento de alguna herramienta, o la aplicación de técnicas, entre otras cosas. Identificarlas y hablar de ellas, permite encontrar soluciones.

El tamaño del proyecto

¿Cuántos meses dura el proyecto?, ¿Estamos a tiempo?

Ser claros en lo que hay que dar.

Los proyectos tienen restricciones de tiempo, alcance, presupuesto. Y tienen parámetros esperados de calidad y funcionalidad. Saber que es lo mas y menos importante para el equipo.

Mostrar que cosas costará.

Tiempo diario a dar, costo de inversión y el tipo de equipo que se necesita para hacer que las cosas funcionen.

Todas estas preguntas y dudas, son sólo el principio, este Deck permite clarificar detalles antes de empezar e incluso abre las puertas a otros tipos de preguntas que ayudaran mucho más, con el fin de tener a todos felices, clientes y desarrolladores.

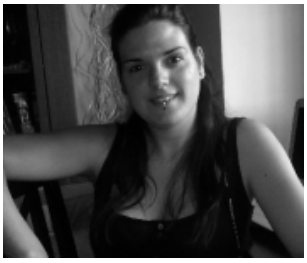
> *La sinceridad al principio del proyecto, es factor clave para su éxito*

¿El enlace de la imagen no funciona? Copia y pega esta URL: http://www.defectivebydesign.org

NgiNX instalado y... ¿ahora qué?

En el número pasado vimos como instalar el servidor web NGiNX, utilizando un enjaulado para añadir seguridad. En este artículo veremos cómo levantar una aplicación php, PhpMyAdmin.

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



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

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

Redes sociales:
Twitter: **@MMontesDiaz**

Una vez instalado nuestro servidor NGiNX, php-fpm y MariaDB, vamos a añadir aplicaciones. Veremos cómo traducir los rewrite a NGiNX, dado que los .htaccess no son tratados por él.

Antes de continuar, comentar que en la última actualización del paquete de NGiNX se ha cambiado el binario de /usr/sbin a /usr/bin y, además está corregido el bug CVE-2013-2028[6], con lo que es conveniente actualizar.

Lo primero que vamos a hacer es instalar los paquetes necesarios:

```
# pacman -S phpmyadmin php-mcrypt
```

Activamos el módulo de php:

```
# echo “extensions = mycrypt” > /etc/php/conf.d/mycrypt
```

Ahora, necesitamos que estos sean vistos desde NGiNX. Para esto podemos optar por dos opciones:

1.- Copiar los archivos de la aplicación.

6 http://mailman.nginx.org/pipermail/nginx-announce/2013/000112.html

2.- Montar los directorios en los que se encuentra instalados en la jaula.

Con la segunda opción tenemos la ventaja de no duplicar espacio y, además, al actualizar el sistema, no necesitaremos intervención manual, salvo que queramos añadir seguridad adicional cambiando los permisos de los archivos, que, una vez actualicemos, deberos volver a establecerlos.

Para hacer esto, editamos el */etc/fstab* y le añadimos lo siguiente:

```
# Para montar los dipostivos que utilizará NGiNX
mount -t tmpfs none /srv/nginx/run -o 'noexec,size=1M'
mount -t tmpfs none /srv/nginx/tmp -o 'noexec,size=100M'

# Montamos los directorios de phpmyadmin
/usr/share/webapps /srv/nginx/usr/share/webapps   none   bind   0   0
/etc/webapps       /srv/nginx/etc/webapps         none   bind   0   0
```

Creamos los puntos de montaje y montamos:

```
# mkdir -p /srv/nginx/etc/webapps
# mkdir -p /srv/nginx/usr/share/webapps
# mount -a
```

Ya está nuestra aplicación, ahora vamos a configurar NGiNX. Editamos el fichero de configuración */srv/nginx/etc/nginx/nginx.conf* y eliminamos todas las referencias a *server*. Al final del archivo, añadimos dónde estará la configuración de los diferentes hosts virtuales:

```
worker_processes  auto;

error_log  /var/log/nginx/error.log;

events {
    worker_connections  1024;
}


http {
    include       mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
    keepalive_timeout  65;
```

```
    #gzip  on;

    #Para configuraciones adicionales,
    #los archivos deben terminar en .conf
    include server.d/*.conf;

    #Para definir los diferentes servidores virtuales
    include server.d/*;
}
```

Creamos un enlace a */usr/share/webapps/phpMyAdmin*:

```
# cd /srv/nginx/srv/html
# ln -s /usr/share/webapps/phpMyAdmin phpmyadmin
```

Creamos los directorios */srv/nginx/etc/nginx/{conf.d ,server.d}* y, dentro de *server.d*, el archivo *default* con el siguiente contenido:

```
 server {
     listen 80;

     root /srv/html;
     index index.html index.htm index.php;

     error_page   500 502 503 504  /50x.html;

     location = /favicon.ico {
        log_not_found off;
        access_log off;
     }

     location = /robots.txt {
        allow all;
        log_not_found off;
        access_log off;
     }

     location ~* \.(jpg|jpeg|gif|png|css|js|ico|xml)$ {
        access_log        off;
        log_not_found     off;
        expires           360d;
     }


     # Userdir, acceder al direcotrio public_html
     # de los usuarios mediante http://<nombrehost>/~<nombreusuario>
     location ~ ^/~(.+?)(/.*)?$ {
         #autoindex on;
         autoindex_exact_size off;
         autoindex_localtime on;
         alias /home/$1/public_html$2;
     }

     #
     # PhpMyAdmin - php
```

```
      #
      location /phpmyadmin/libraries {
           deny all;
      }
      location /phpmyadmin/setup/lib {
           deny all;
      }

      location /phpmyadmin {
           try_files $uri $uri/ @rewrite ;
      }

      location ~ phpmyadmin(.+\.php)$ {
           try_files $uri =404;
           fastcgi_pass   127.0.0.1:9000;
           fastcgi_param PHP_ADMIN_VALUE
open_basedir="/tmp/:/srv/html/:/etc/webapps/:/usr/share/webapps/:./";
           fastcgi_index  index.php;
       include        fastcgi.conf;
      }

      #
      #  php por defecto
      #
      location ~ (.+\.php)$ {
           fastcgi_pass 127.0.0.1:9000;
           fastcgi_index  index.php;
           fastcgi_param  PHP_ADMIN_VALUE  open_basedir="/srv/html/:/tmp/:./";
           include        fastcgi.conf;
      }

      location / {
       allow all;
       try_files $uri $uri/  @rewrite ;
      }

      #
      # Acceso al Status de NGiNX
      #
      location /nginx_status {
           #Ejemplo para permitir el acceso sólo
           #a nuestra red local
           deny 192.168.0.1 ;
           allow 192.168.0.0/24 ;
           allow 127.0.0.1 ;
           deny all ;
           stub_status on;
      }

      location @rewrite {
             rewrite ^ /index.php last;
      }

      #
      #limitando archivos ocultos, .bak, .old
      #
      location ~* (/\.|\.bak$|\~$|\.old$) {
           access_log off;
           log_not_found off;
           deny all;
      }
  }
```

Todavía nos falta un pequeño detalle más para que la aplicación phpMyAdmin funcione correctamente: Indicar en su archivo de configuración, */etc/webapps/phpmyadmin/config.inc.php*, dónde se encuentra la BBDD. Lo editamos y establecemos como host **127.0.0.1** en lugar de **localhost,** aunque lo ideal es que la BBDD se ejecute en otro equipo:

```
...
/* Authentication type */
$cfg['Servers'][$i]['auth_type'] = 'cookie';
/* Server parameters */
$cfg['Servers'][$i]['host'] = '127.0.0.1';
$cfg['Servers'][$i]['connect_type'] = 'tcp';
...
```

Ahora, reiniciamos nginx y php-fpm:

```
#sudo systemctl restart php-fpm nginx-jail
```

Para acceder a phpMyAdmin, bastará ir al navegador y dirigirnos a: http://localhost/phpmyadmin

Consejo: Si quieren activar la configuración avanzada de phpMyAdmin, visiten https://wiki.archlinux.org/index.php/PhpMyAdmin, donde encontrarán una explicación detallada para hacerlo.

Para incluir otro ejemplo, voy a explicar cómo instalar Friendica[7]. Esta aplicación es una red social distribuida, muy fácil de instalar y con características interesantes: Varios perfiles por cuenta, posibilidad de cambiar una cuenta de un servidor a otro, federada con otras redes (por ejemplo StatusNet y Diáspora), posibilidad de seguir RSS, etc.

Para instalar la aplicación, necesitaremos el paquete git:

```
# pacman -S git
# cd /srv/nginx/srv/html/
# git clone https://github.com/friendica/friendica.git friendica
# git clone https://github.com/friendica/friendica-addons friendica/addon
```

Instalamos las extensiones php que nos faltan:

```
# pacman -S php-curl php-gd php-intl
# echo "extension=iconv.so"   > /etc/php/conf.d/iconv.ini
# echo "extension=imap.so"    > /etc/php/conf.d/imap.ini
# echo "extension=gd.so"      > /etc/php/conf.d/gd.ini
```

7 http://friendica.com/

```
# echo “extension=openssl.so” > /etc/php/conf.d/openssl.ini
```

Necesitamos poder enviar correos para poder enviar las notificaciones y para el registro de usuarios. Para ello vamos a utilizar msmtp[8], que nos proporciona compatibilidad con sendmail y nos permite utilizar una cuenta de correo, como por ejemplo, Gmail o Yahoo! para enviar los correos. De esta manera no necesitamos levantar un servicio como Postfix, que se escapa un poco de la temática del artículo.

```
# pacman -S msmtp-mta
```

Ahora, creamos el archivo /etc/msmtprc y configuramos nuestra cuenta:

```
defaults
auth            on
tls             on
tls_trust_file   /usr/share/ca-certificates/mozilla/Equifax_Secure_CA.crt

account        gmail
host           smtp.gmail.com
port           587
from           username@gmail.com
user           username@gmail.com
password       password

account        yahoo
tls_trust_file /usr/share/ca-ertificates/mozilla/Thawte_Premium_Server_CA.crt
host           smtp.correo.yahoo.es
port           25
from           username@yahoo.es
user           username@yahoo.es
password       password

# Cuenta por defecto
account default : gmail
```

Creamos el fichero de configuración:

```
# touch /srv/nginx/srv/html/friendica/.htconfig.php
# chown http:http /srv/nginx/srv/html/friendica/.htconfig.php
```

Una vez hecho esto, vamos a configurar el servidor virtual. Creamos el archivo */svr/nginx/etc/nginx/server.d/<nombre_del_dominio>*, para el ejemplo *frndc.archnifa.org*, y le añadimos lo siguiente:

8 https://wiki.archlinux.org/index.php/Msmtp

```
server {
     listen 80;
     listen 443 ssl;
     server_name frndc.archninfa.org;

     root /srv/html/friendica;
     index index.php;

     access_log /var/log/nginx/frndc.archnifa.org.access;
     error_log /var/log/nginx/frndc.archnifa.org.error;

     server_name_in_redirect on;

     location = /favicon.ico {
        log_not_found off;
        access_log off;
     }

     location = /robots.txt {
        allow all;
        log_not_found off;
        access_log off;
     }

     location ~* \.(jpg|jpeg|gif|png|css|js|ico|xml)$ {
        access_log        off;
        log_not_found     off;
        expires           360d;
     }


     # NO utilizar if para los rewrite, en lugar de esto:
     #location / {
     #      if (!-e $request_filename){ rewrite ^(.*)$ /index.php?q=$1; }
     #}
     # Es mejor utlizar try_files:

     location / {
          try_files $uri $uri/ @rewrite;
     }

     location ~ (.+\.php)$ {
          try_files $uri =404;
          fastcgi_param HTTPS on;
          fastcgi_pass   127.0.0.1:9000;
          fastcgi_param PHP_ADMIN_VALUE
             open_basedir="/tmp/:/srv/html/friendica/:./:/srv/http/friendica/";
          fastcgi_index  index.php;
      include        fastcgi.conf;
     }

     # Para centralizar mejor los rewrite, derivamos con try_files
     # a @rewrite y, allí, los especificamos:
     location @rewrite {
             rewrite ^(.*)$ /index.php?q=$1;
     }

     #
     #limitando archivos ocultos, .bak, .old
     #
     location ~* (/\.|\.bak$|\~$|\.old$) {
          access_log off;
          log_not_found off;
          deny all;
```

```
    }
}
```

Si estamos haciendo pruebas en nuestra lan y no tenemos un servidor de DNS, no olvidar añadir el dominio tanto al */etc/hosts*, como al */srv/nginx/etc/hosts*.

Debemos crear una BBDD vacía y un usuario con permisos de escritura para la misma, podemos utilizar para esto phpMyAdmin.

Damos permisos a carpetas específicas, por exigencia de la aplicación:

```
# chown http:http /srv/html/friendica/view/smarty3
```

Creamos los certificados:

```
# mkdir /srv/nginx/etc/nginx/ssl
# cd /srv/nginx/etc/nginx/ssl
# openssl genrsa -des3 -out server.key 1024
# openssl req -new -key server.key -out server.csr
# cp server.key server.key.org
# openssl rsa -in server.key.org -out server.key
# openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
# cp -r /etc/ssl /srv/nginx/etc/
```

Creamos el archivo */srv/nginx/etc/nginx/conf.d/ssl.conf* con el siguiente contenido:

```
#Configuración de SSL:
ssl_protocols SSLv3 TLSv1;
ssl_prefer_server_ciphers on;
ssl_certificate      /etc/nginx/ssl/server.crt;
ssl_certificate_key  /etc/nginx/ssl/server.key;

# Prevenir ataques BEAST y otros exploits
ssl_session_cache shared:SSL:10m;
ssl_session_timeout 5m;
ssl_ciphers             ECDHE-RSA-AES256-SHA384:AES256-SHA256:RC4:HIGH:!MD5:!
aNULL:!EDH:!AESGCM;
add_header              Strict-Transport-Security max-age=500;
ssl_ecdh_curve          secp521r1;
```

Reiniciamos el servidor y nos dirigimos con nuestro navegador a http://frnc.archinfa.org:

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

Conclusión

Cómo hemos podido ver, aunque tengamos la posibilidad de utilizar múltiples cláusas

"root" dentro del contexto server, lo ideal es utilizar sólo una. Igual ocurre con la cláusula index, resultará más clara revisión de la configuración.

Con respecto a los *rewrite* y adaptación de los *.htaccess,* debemos tener en cuenta que, aunque herramientas como http://winginx.com/htaccess, utilicen *if* para crearla, no es lo correcto. El uso de *if* para estos casos puede provocar diversos problemas, de ahí la utilización de *try_files*. En el ejemplo utilizado, un *.htaccess* similar a:

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php?q=$1 [E=REMOTE_USER:%{HTTP:Authorization},L,QSA]
```

Se traduce en:

```
     location / {
          try_files $uri $uri/ @rewrite;
     }

.................................................

     location @rewrite {
             rewrite ^(.*)$ /index.php?q=$1;
     }
```

Enlaces de interés:

http://wiki.nginx.org/Main

http://wiki.nginx.org/IfIsEvil

http://wiki.nginx.org/Pitfalls

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

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

http://friendica.com/

http://winginx.com/htaccess

El olvidado mundo de las variables en PHP

Variables locales; variables globales; variables súper globales; variables estáticas; variables variables; variables constantes... ¿de verdad tienes la seguridad de conocerlas todas?

Escrito por: **Eugenia Bahit** (GLAMP Hacker & eXtreme Programmer)

Eugenia es **Arquitecta de Software**, docente e 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**, **The Linux Foundation** y **Debian Hackers**. Creadora de python-printr, Europio Engine y colaboradora de Vim.

Webs:
Cursos de programación: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

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

Frecuentemente me encuentro con que una gran cantidad de programadores se refiere al ámbito de las variables de manera equivocada, confundiendo a veces, ámbito con tipo. Me pareció una buena idea centrarme en este punto con la esperanza de dejar el tema más claro.

Sobre el ámbito de una variable

Para comenzar, lo primero que debemos dejar en *"blanco sobre negro"* es **a qué nos referimos realmente cuando hablamos del ámbito de una variable**. Básicamente, el ámbito al que pertenece una variable **se refiere a la disponibilidad de dicha variable** en el contexto de la aplicación.

Una variable puede estar disponible a nivel global o local. PHP, también incorpora el concepto de variable súper global. Veamos a qué se refiere cada una.

Variable global

Son variables accesibles desde cualquier parte de la aplicación, independientemente del tipo de estructura de control del que se trate. Las variables globales son aquellas que el

programador, define con la palabra `global` delante del nombre de la variable.

Si una variable global se define dentro de una función, ésta deberá ser ejecutada previamente para que la variable esté disponible a nivel global:

```
function foo() {
    global $bar;
    $bar = 75;
}

# Esto fallará:
print $bar;

# Sin embargo, al llamar a foo(), $bar estará disponible fuera de la función:
foo();
print $bar;
```

Lo mismo sucede, si la variable global es definida dentro de un método de clase:

```
class Foo {

    static function bar() {
          global $bar;
          $bar = 75;
    }

}

Foo::bar();
print $bar;
```

Sin embargo, cualquier variable definida fuera de una función o método de clase, estará disponible a nivel global sin necesidad de ser declarada como `global`.

```
/*
Archivo foo.php
   Tanto $foo como $i son variables globales.
   Los cambios realizados dentro del bucle while, también aplican a nivel global.
*/

$foo = 75;

$i = 0;
while($i < 5) {
    $foo--;
    $i++;
}

Archivo bar.php
require_once 'foo.php';
print $foo;  # Salida: 70
```

Pero, si se quisiera acceder a `$foo` o a `$i` desde una función o método de clase, previamente deberá ser llamada con `global`:

```
Archivo foo.php
$foo = 75;

$i = 0;
while($i < 5) {
    $foo--;
    $i++;
}


Archivo bar.php
require_once 'foo.php';

function bar() {
    # Esto fallará:
    print $foo;

    # Pero llamando a global funcionará:
    global $foo;
    print $foo;
}

bar();
# Salida: 70
```

Vale aclarar que una variable global definida por el programador, deberá ser incluida en el *script* para que esté disponible. Vale decir que si en el archivo `foo.php` defino una variable `$foo` a la que deseo acceder desde el archivo `bar.php`, en este último, debo incluir al archivo `foo.php`.

Variable local

Las variables locales son aquellas que se definen dentro de una función o método de clase, sin anteceder la palabra global. Este tipo de variables, solo serán accesibles desde la función o método que las ha declarado.

```
function foo() {
    $bar = 75;
}

# Esto fallará ya que $bar solo está disponible dentro de foo()
print $bar;


class Foo {

    static function bar() {
          $bar = 75;
    }

}

# Esto también fallará ya que $bar solo está disponible dentro de Foo::bar()
print $bar;
```

Los parámetros de una función o método de clase también son variables locales:

```
function foo($bar=75) {
    # Esto funciona:
    print $bar;
}


# Pero esto, fala:
print $bar;
```

> *Toda variable definida dentro de una función o método de clase, será de ámbito local si no es declarada como global*

Variables súper globales

Las variables súper globales son -a nivel ámbito- variables globales y se diferencian de éstas en que:

- No son definidas por el programador, sino que vienen incorporadas al lenguaje;
- No necesitan ser llamadas mediante `global`;

Entre las variables súper globales más usuales, nos podremos encontrar a:

```
array $_POST
array $_FILES
array $_GET
array $_SERVER
array $_COOKIE
array $_SESSION
array $http_response_header
array $argv
int $argc
```

Sobre el tipo de las variables

Cuando hablamos de "tipo de variables" es necesario aclarar que no estaos haciendo referencia al tipo de datos de la variable sino, al "modo" de la variable.

Existen 4 tipos básicos de variables:

1. **Variables:**

   las convencionales :)

2. **Variables estáticas:**

aquellas variables locales que conservan su valor tras la ejecución de la función o método que las declara.

3. **Variables variables:**

   aquellas cuyo nombre de variable se define en tiempo de ejecución.

4. **Constantes:**

   similares a las variables de ámbito global con la gran diferencia de que poseen un valor "constante" (no varía)

```
function foo() {
    $variable_local = 'bar';
    global $variable_global;
    $$variable_local = 'variable variable cuyo nombre es $bar';
}

const CONSTANTE = 0;
```

Variables estáticas

Una variable estática es de ámbito local pero se diferencia de una variable tradicional por el hecho de que conserva su valor tras la ejecución del ámbito.

Mientras que una variable local desaparece tras la ejecución del ámbito, provoca que su valor se restablezca en cada llamada:

```
function foo() {
    $bar = 1;
    $bar++;
    print $bar;
}

foo();
# Salida: 2

foo();
# Salida: 2
```

Sin embargo, al declararla como estática, conserva su valor:

```
function foo() {
    static $bar;
    $bar++;
    print $bar;
}
```

```
foo();
# Salida: 1

foo();
# Salida: 2

foo();
# Salida: 3

foo();
# Salida: 4
```

Como se puede apreciar, en cada llamada, la variable estática `$bar` ha conservado su último valor.

Variables variables

Las variables variables son una forma de definir variables -locales o globales- en tiempo de ejecución y resultan ideales en instrucciones iterativas. Son variables cuyo nombre, se define en tiempo de ejecución dejando disponible una nueva variable.

```
$nombre = 'foo';
$$nombre = 75;  # esta variable se llama $foo
print $foo;
# Salida: 75
```

Un uso frecuente de variables variables se da en estructuras iterativas. Un clásico ejemplo es la definición de variables para los datos enviados desde un formulario.

La forma típica de definición sería:

```
$nombre = $_POST['nombre'];
$apellido = $_POST['apellido'];
$email = $_POST['email'];
$cumpleanios = $_POST['cumpleanios'];
$dni = $_POST['dni'];
$nacionalidad = $_POST['nacionalida'];
```

Sin embargo, una forma más corta, se logra mediante el uso de variables variables:

```
foreach($_POST as $key=>$value) $$key = $value;
```

Lo anterior, dejará disponibles las siguientes variables:

```
$nombre          $apellido          $email       $cumpleanios
$dni             $nacionalidad
```

Constantes

Hasta antes de la versión 5.3 de PHP, una constante se definía con la función `define()`.

```
define('FOO', 1);
print FOO;  # Salida: 1

function bar() {
    print FOO;
}

bar();  # Salida: 1
```

En PHP 5.3 se incorpora la definición de constantes de ámbito global mediante **`const`** extendiendo así dicha instrucción que en la versión 5.2 solo se encontraba disponible en el ámbito de una clase.

```
const FOO = 1;
```

Mientras que una constante definida mediante `define()` podía recibir una expresión como valor:

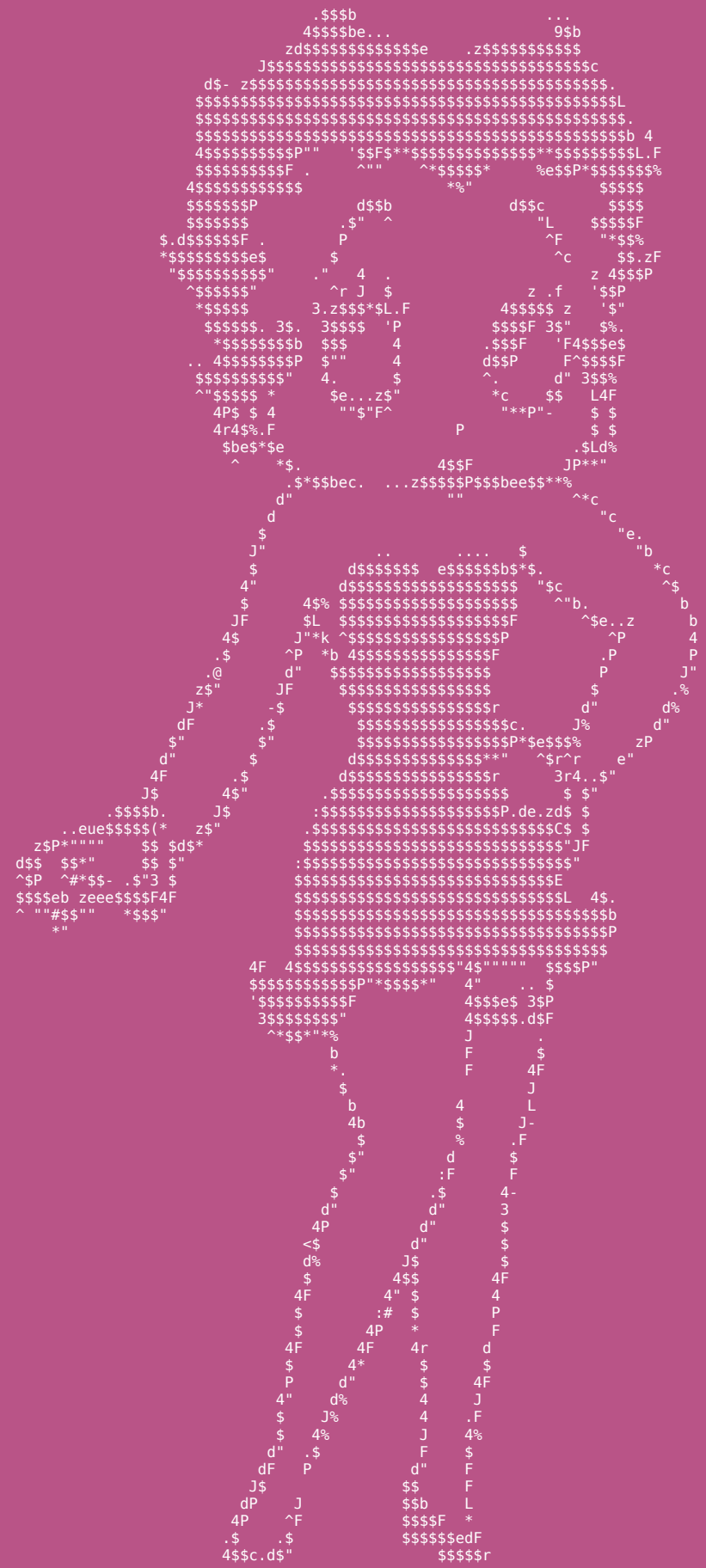
```
$bar = 12;
define('FOO', $bar);
print FOO;  # Salida: 12

define('BAR', $bar * 2 / 3);
print BAR;  # Salida: 8
```

Una constante definida mediante const, solo puede recibir valores directos pero no expresiones:

```
# CORRECTO
const FOO = 1;
const BAR = FOO;

# INCORRECTO: Los siguientes casos, fallan:
$bar = 15;
const BAR2 = $bar;
const FOOBAR = 5 * 3 / 2;
```



Betty Boop

(Anónimo)

http://www.flipmytext.com/ascii/bettyboop.php

U! Tu zona exclusiva

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

Te presentamos **gulBACzine**, un nuevo Magazine sobre Software Libre

Desde Hackers & Developers Magazine, queremos invitarlos a todos a conocer a **gulBACzine:** Un nuevo magazine para usuarios de Software Libre, dirigido por el lug **gulBAC** de Argentina.

El primer número de la revista se encuentra disponible para la descarga en PDF desde el pasado 20 de Mayo de 2013 en **http://www.gulbac.org.ar/gulbaczine**

Próximos eventos:

PyConES 2013

23 y 24 de Noviembre · Madrid, España.

http://es.pycon.org

Se dice en Twitter...

@LeanMiguel27: @HackDevMagazine los quiero felicitar por lo q hacen, un gran trabajo!! es de gran ayuda y utilidad! me encanta! saludos chicas.

@VR1387: Los articulos que tiene http://www.hdmagazine.org/ son excelentes (...)

@espiritullama: @HackDevMagazine @MMontesDiaz Que genial, un artículo de MariaDB escrito por María jeje

@kardenas3: @HackDevMagazine que buena se la ve a la revista recien me descargue el primer numero, lo descubri por taringa

@dontachoo: no puede ser! me he perdido de los últimos ejemplares de @HackDevMagazine // en fin de semana los repaso todos :)

¡GRACIAS POR LEERNOS!