Tips, Tricks & Hacks en PHP

PARA SUS VERSIONES 5.3, 5.4 Y 5.5

Eugenia Bahit

Sígueme

Compilado de distribución Libre y Gratuita

www.originalhacker.org

Índice de Contenidos de la Edición Especial Nº11 de The Original Hacker

por Eugenia Bahit

Refactoring

Buenas prácticas

Con una sola línea de código

Hacks Experimentales

Manipulación de Archivos

Programación Orientada a Objetos

Inteligencia Artificial

MVC

Bases de Datos

Los **«Europio Engine HOWTO»** podrás encontrarlos a partir de **página 32**

#00:00 (truco) Refactoring

Desempaquetado de parámetros para sustituir el exceso de argumentos en funciones

Suele suceder que una función se nos llena de parámetros (argumentos) y resulta sumamente engorroso. Un ejemplo podría verse como el siguiente:

```
function foo($nombre, $apellido, $fecha_nacimiento, $nacionalida, $sexo, $pais,
    $ciudad, $cp) {
    print "$apellido, $nombre ($nacionalida)";
    # ...
}
```

Para evitarlo, el truco consiste en definir un único argumento y utilizar la función `extract()` para obtener todas las variables a partir de un array asociativo:

```
function foo($kwargs) {
    extract($kwargs);
    print "$apellido, $nombre ($nacionalida)";
    # ...
}
```

Al invocar a la función se le pasa un array asociativo:

```
$data = array(
    "nombre"=>"Juan",
    "apellido"=>"Pérez",
    "fecha_nacimiento"=>"23/04/2000",
    "nacionalidad"=>"uruguaya",
    "sexo"=>"M",
    "pais"=>"Argentina",
    "ciudad"=>"Lanús",
    "cp"=>1824
);

foo($data);
```

Las claves del array serán usadas como nombres de variables.

#01:00 (truco) Refactoring

Funciones que solo definen variables temporales para evitar códigos espagueti

Nuevamente, empleando `extract()` podemos evitar funciones kilométricas, aplicando una de las técnicas de *refactoring* más usadas: la extracción de código.

El truco consiste en retornar dichas variables sin necesidad de generar un *array* manualmente. Para ello, se crea una función que solo se encargue de definir las variables temporales retornándolas mediante `get_defined_vars()`:

```
function set_vars() {
    $nombre = "Juan";
    $apellido = "Pérez";
    $fecha_nacimiento = "11/11/2011";
    $nacionalida = "argentina";
    $sexo = "M";
    $pais = "Argentina";
    $ciudad = "Junín";
    $cp = "1111";
    return get_defined_vars();
}
```

En la función *refactorizada* (desde la cual se extrajo el código), se llama a la nueva función, extrayendo las variables que ésta retorna:

```
function refactorizada() {
    extract(set_vars());
    print $apellido . ', ' . $nombre;
    # ...
}
```


#02:00 (tip) Refactoring

¿Defines muchas variables con el mismo valor dependiendo de una evaluación condicional o más?

Si te encuentras en una situación similar a la siguiente:

```
if(condicion) {
    $uno = $variable * 45;
    $dos = file_get_contents('foo.php');
    $tres = str_replace(' ', '-', 'hola mundo');
    $cuatro = array(1, 2, 3);
} else {
    $uno = "datos insuficientes";
    $dos = "datos insuficientes";
    $tres = "datos insuficientes";
    $cuatro = "datos insuficientes";
}
```

La lógica indica que las variables deberían tener inicialmente el mismo valor y sobrescribirse en caso que la condición se cumpla:

```
$uno = "datos insuficientes";
$dos = "datos insuficientes";
$tres = "datos insuficientes";
$cuatro = "datos insuficientes";

if(condicion) {
    $uno = $variable * 45;
    $dos = file_get_contents('foo.php');
    $tres = str_replace(' ', '-', 'hola mundo');
    $cuatro = array(1, 2, 3);
}
```

Sin embargo, existe una mejor forma de cumplir con las consecuencias lógicas empleando mejores prácticas, como la asignación múltiple de variables:

```
list($uno, $dos, $tres, $cuatro) = array_fill(0, 4, "datos insuficientes");

if(condicion) {
    $uno = $variable * 45;
    $dos = file_get_contents('foo.php');
    $tres = str_replace(' ', '-', 'hola mundo');
    $cuatro = array(1, 2, 3);
}
```


Cuando el mismo escenario se presenta pero debiendo evaluar una condición para cada variable, otra buena alternativa suele ser la siguiente:

```
list($uno, $dos, $tres, $cuatro) = array_fill(0, 4, "datos insuficientes");

if(condicion 1) $uno = $variable * 45;
if(condicion 2) $dos = file_get_contents('foo.php');
if(condicion 3) $tres = str_replace(' ', '-', 'hola mundo');
if(condicion 4) $cuatro = array(1, 2, 3);
```


#03:00 (tip) Buenas prácticas

Condicionales ternarios para evitar grandes bloques if/else

Frente a un escenario en el cuál gran cantidad de variables se definirán según los resultados booleanos de una única evaluación condicional como en el siguiente caso:

```
if(condicion) {
    $variable = valor;
} else {
    $variable = otro valor;
}
```

Resulta más legible implementar condicionales ternarios:

```
$variable = (condicion) ? valor : otro valor;
```

Para saber: los condicionales ternarios no se limitan solo a la definición de variables. Pueden emplearse en conjunción con muchas otras instrucciones:

```
return (condicion) ? valor si verdadero : valor si falso;
print (condicion) ? valor si verdadero : valor si falso;
(condicion) ? accion si verdadero : accion si falso;
```

Un ejemplo con el antes y el después:

```
ANTES:
$bruto = 100;
if($bruto < 100) {
    $neto = $bruto;
} else {
    $neto = $bruto * 0.9;
}

DESPUÉS:
$bruto = 100;
$neto = ($bruto < 100) ? $bruto : $bruto * 0.90;
```


#04:00 (tip & trick) Buenas prácticas

Mostrar errores en desarrollo y ocultarlos en producción de forma dinámica

Ver los errores en pantalla por supuesto que es muy útil. Pero ya sabemos que es una muy mala política de seguridad mostrarlos en producción. Alternar entre mostrarlos u ocultarlos es tan simple como definir una constante y cambiar su valor de `true` a `false` dependiendo de si se está en producción o no.

```
const PRODUCTION = false;  # true cuando se lleve el código a producción
```

Para que el reporte de errores en desarrollo sea completo, independientemente de cómo esté configurado en el php.ini, pueden emplearse las siguientes instrucciones:

```
if(!PRODUCTION) {
    ini_set('error_reporting', E_ALL | E_NOTICE | E_STRICT);
    ini_set('display_errors', '1');
    ini_set('track_errors', 'On');
} else {
    ini_set('display_errors', '0');
}
```

#05:00 (tip) Buenas prácticas

Evitar errores con las rutas de importación al hacer un include o require

Solo debes establecer la ruta de inclusión a la carpeta raíz de la app e indicar todas las rutas en includes y requires desde ésta:

```
ini_set('include_path', $_SERVER['DOCUMENT_ROOT']);
```


#06:00 (hack) Soluciones rápidas

Completar una contraseña de forma aleatoria hasta alcanzar la longitud esperada

¿El usuario dejó la contraseña en blanco? ¿utilizó pocos caracteres? En una sola instrucción, la modificamos, generando una clave aleatoria segura y sin hacer absolutamente ningún esfuerzo de validación. Solo hay que emplear funciones anónimas y variables por referencia.

Generación de contraseñas aleatorias seguras con una sola línea de código:

```
$rnd = function(&$a, $n=8) { while(strlen($a) < $n) $a .= chr(rand(33, 126)); };
```

Ejemplo de uso:

```
$rnd($_POST['clave']);      # Modifica el valor de $_POST['clave']
$rnd($_POST['clave'], 24);  # Clave de 24 caracteres
```

El *hack* en acción:

```
php > $rnd = function(&$a, $n=8) { while(strlen($a) < $n) $a .= chr(rand(33, 126)); };
php >
php > $rnd($_POST['clave']);
php > print $_POST['clave'];
&PSb[Y]K
php >
php > $rnd($_POST['clave'], 12);
php > print $_POST['clave'];
&PSb[Y]K9sCr
php > $rnd($_POST['clave'], 32);
php > print $_POST['clave'];
&PSb[Y]K9sCrih=QfQR]iUB3%QP`W:x]
```

Importante: no olvides *hashear* las contraseñas antes de guardarlas:

```
md5($_POST['clave']);
```


#07:00 (truco) Soluciones rápidas

Todos los campos del array $_POST a variables, limpios y saneados en un solo paso

Con la ayuda de array_map podemos recorrer un array aplicando una misma función a cada uno de sus elementos. Con este sencillo truco, en a penas un paso, limpiamos los campos de un formulario para luego extraerlos en variables:

```
function clean($str) { return htmlentities(strip_tags($str), ENT_QUOTES); }
extract(array_map('clean', $_POST));
```

Ejemplo en funcionamiento:

```
php > $_POST['a1'] = '<b>hola</b> mundo';
php > $_POST['a2'] = 'mundo <script>alert();</script>';
php > $_POST['a3'] = '"hola\' mundo"';
php > function clean($str) { return htmlentities(strip_tags($str), ENT_QUOTES); }
php > extract(array_map('clean', $_POST));
php > print $a1;
hola mundo
php > print $a2;
mundo alert();
php > print $a3;
&quot;hola&#039; mundo&quot;
```

#08:00 (truco) Soluciones rápidas

Importar todos los archivos de una carpeta

Si eres de los que coloca los archivos PHP organizados dentro de directorios, un buen truco para incluirlos de forma automática y despreocuparte de agregar nuevas instrucciones *include* o *require* es el siguiente:

```
$c = "/path/to/folder/";
foreach(scandir($c) as $f) if(is_file($c.$f)) include_once $c.$f;
```


#09:00 (tip, trick & hack) Experimentales

Wrappers y decoradores como en Python

El siguiente código, emula un decorador de restricción de acceso en Python:

```
$decorator = function($func) {

    $wrapper = function() use ($func) {
        $logueado = isset($_SESSION['login']) ?  $_SESSION['login'] : False;
        if($logueado) {
            return call_user_func_array($func, func_get_args($func));
        } else {
            return "Debe loguearse";
        }
    };

    return $wrapper;
};
```

Dada una función candidata a ser «decorada»:

```
function foo($arg1, $arg2...) {
    # código de la función
}
```

El **hack** para lograrlo, consiste en retornarla desde un *closure* envuelto a la vez, por el decorador:

```
$foo = $decorator(
    function($arg1, $arg2...) {
        return guardar($arg1, $arg2...);
    }
);
```

Finalmente se debe llamar al *closure* en vez de llamar a la función original:

```
$foo(1, array(3, 4)...);
```

Tip: Conviene definir todas las funciones normalmente y colocar el decorador a nivel del *core*. Luego, a toda función a la cual se le quiera restringir el acceso (es decir, requiera que un usuario se encuentre *logueado* en el sistema), se le crea el *closure*. Al momento de

efectuar la llamada de la función, si se la tiene que invocar de forma dinámica y no se sabe si es o no restringida, se puede emplear el siguiente **truco** utilizando «variables variables»:

```
$funcion = 'guardar';

if(isset($$funcion)) {  # verifica si hay una variable definida con el nombre de la función
    $$funcion($argumentos);  # llama al closure
} else {
    $funcion($argumentos);  # llama a la función riginal normalmente
}
```

Ejemplo en funcionamiento:

```
function modificar($id) {
     // código de la función privada …
     return "Registro con ID $id se ha actualizado" . chr(10);
}


function eliminar($id) {
     // código de la función privada …
     return "Registro con ID $id se ha eliminado" . chr(10);
}


function ver($id) {
     // código de la función pública …
     return "Estoy mostrando los datos del registro $id" . chr(10);
}


# Envolturas
$modificar = $decorator(function($id) { return modificar($id); });
$eliminar = $decorator(function($id) { return eliminar($id); });


# ******* pruebas *******
session_start();

$_SESSION['loggin'] = False;
$funcion = 'modificar';
$arg = 15;

if(isset($$funcion)) {
    $salida = $$funcion($arg);
} else {
    $salida = $funcion($arg);
}
print $salida;
// Debe loguearse

$_SESSION['loggin'] = False;
$funcion = 'ver';
$arg = 15;

if(isset($$funcion)) {
    $salida = $$funcion($arg);
} else {
    $salida = $funcion($arg);
}
```

```
print $salida;
// Estoy mostrando los datos del registro 15


$_SESSION['loggin'] = True;
$funcion = 'eliminar';
$arg = 15;

if(isset($$funcion)) {
    $salida = $$funcion($arg);
} else {
    $salida = $funcion($arg);
}
print $salida;
// Registro con ID 15 se ha eliminado
```


#10:00 (hack) Manipulación de archivos

Mostrar archivos binarios desde directorios no accesibles por el navegador

Cuando se trabaja con sistemas de carga de archivos, una buena política de seguridad es almacenarlos en un directorio que no se encuentre servido (es decir, que no sea accesible por el navegador) puesto que el mismo deberá contar con permisos de escritura.

Cuando los archivos que se cargan son binarios que deben ser mostrados estáticamente desde código HTML -o descargados-, es necesario que el navegador tenga acceso a ellos.

Para mostrarlos (o permitir su descarga) sin tener que habilitar un directorio servido con permisos de escritura, el *hack* consiste en servir los archivos mediante PHP, leyéndolos en modo binario:

```
# Archivo: file.php

$archivo = "/path/to/private/folder/{$_GET['f']}";

if(file_exists($archivo)) {
     $finfo = finfo_open(FILEINFO_MIME_TYPE);
     $mime = finfo_file($finfo, $archivo);
     finfo_close($finfo);
     header("Content-Type: $mime");
     readfile($archivo);
}
```

Luego, en los archivos HTML se podrán invocar de la siguiente forma:

```
<img src='file.php?f=imagen.png'>
<a href='file.php?f=archivo.pdf'>Descargar PDF</a>
```


#11:00 (hack) Manipulación de archivos

Mostrar imágenes contenidas en un archivo ZIP no accesible desde el navegador

Este hack implementa el truco anterior para permitir mostrar en HTML, archivos de imágenes que se encuentran dentro de un ZIP.

```
// archivo: image_show.php

$archivo_zip = "imagenes.zip";

if(isset($_GET['filename'])) {
    $imagen = "zip://$archivo_zip#{$_GET['f']}";
    $finfo = finfo_open(FILEINFO_MIME_TYPE);
    $mime = finfo_file($finfo, $imagen);
    finfo_close($finfo);
    header("Content-Type: $mime");
    readfile($imagen);
} else {
    $zipobj = new ZipArchive;
    $recurso = $zipobj->open($archivo_zip);
    $i = 0;
    while($nombre = $zipobj->getNameIndex($i)) {
        print "<img src='image_show.php?f=$nombre'><br>" . chr(10);
        $i++;
    }
    $zipobj->close();
}
```


#12:00 (truco) Orientación a Objetos

Invocar métodos no estáticos de clases instanciables en el mismo paso, en versiones anteriores a 5.4

El truco consiste en crear funciones con el mismo nombre que las clases, que retornen una instancia de la clase:

```
class Foo {

    function bar() {
        print 'soy bar';
    }
}

function Foo() { return new Foo(); }
```

#13:00 (hack) Orientación a Objetos

Truco #13 aplicado a todas las clases (incluidas las built-in)

El hack consiste en tomar el truco anterior y utilizando la función get_declared_clases(), emplearla para crear una función de forma dinámica, por cada clase instanciable del sistema, incluidas las nativas de PHP:

```
$funcion = "
function CLASE() {
    \$args = (func_num_args()) ? func_get_args() : null;
    return @ new CLASE(\$args);
}
";

$clases = get_declared_classes();

foreach($clases as $clase) {
    $alias = str_replace('CLASE', $clase, $funcion);
    eval($alias);
}

print DateTime()->format('c');
// salida: 2014-12-07T16:52:50-03:00
```


#14:00 (hack) Orientación a Objetos

Llamar a cualquier función de PHP con estilo orientado a objetos

Mediante este hack podremos invocar cualquier función nativa de PHP, con el estilo orientado a objetos, gracias al método mágico __call():

```
class PHP {

    function __construct() {
        $funcs = get_defined_functions();
        $this->builtin_functions = $funcs['internal'];
    }

    function __call($func, $args) {
        if(in_array($func, $this->builtin_functions)) {
            return call_user_func_array($func, $args);
        } else {
            print "PHP no posee una función llamada $func";
        }
    }

}

function PHP() { return new PHP(); }
```

Ejemplo de uso:

```
print PHP()->str_replace('Mundo', 'Pepe Grillo', 'Hola Mundo!');
// salida: Hola Pepe Grillo!

PHP()->print_r(array(1, 2, 3));
/*
Salida:
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
*/

PHP()->foo();
// Salida: PHP no posee una función llamada foo
```

#15:00 (hack) Orientación a Objetos

Recuperación de objetos excluyendo propiedades innecesarias

El *hack* consiste en destruir las propiedades del objeto antes de invocar al método get(). Por ejemplo, dada una clase Foo() similar a la siguiente:

```
class Foo {
    function __construct() {
        $this->propiedad_1 = '';
        $this->propiedad_2 = '';
        # ...
        $this->propiedad_7 = '';
        $this->propiedad_8 = '';
        # ...
        $this->propiedad_que_necesito = '';
    }
}
```

Eliminar mediante unset() las propiedades innecesarias antes de llamar a get():

```
$obj = new Foo();
foreach($obj as $property=>$value) {
    if($property != "propiedad_que_necesito") unset($obj->$property);
}
$obj->get();  # $obj solo tendrá la propiedad "propiedad_que_necesito"
```

Esto permitirá ahorrar una innumerable cantidad de recursos cuando un requerimiento -generalmente gráfico- nos demande recuperar solo unas pocas propiedades (un ejemplo típico de ello, es necesitar una lista visual de todos los usuarios del sistema donde solo se muestre el nombre y una ID).

#16:00 (hack) Orientación a Objetos

Guardar datos que no son propiedades del objeto

Nunca conviene llenar un objeto de propiedades con la sola justificación de que es un dato necesario. Si un dato no es una característica del objeto, no debe ser una propiedad del mismo. Sin embargo, si se lo necesita, se puede emplear la inversa del hack anterior para guardarlo.

La tabla en la DB deberá contar previamente con los campos para dichos datos y antes de llamar al método save() del objeto, se crearán las «pseudo propiedades» que serán destruidas luego de haberlas guardado:

```
class Foo {
    function __construct() {
        $this->propiedad = '';
    }
}

$obj = new Foo();
$obj->dato1 = 'valor';
$obj->dato2 = 'valor';
$obj->save();
unset($obj->dato1);
unset($obj->dato2);
```


#17:00 (truco) Inteligencia Artificial

Obtener palabras claves de un bloque de texto

```
$palabras = explode(" ", str_replace(chr(10), ' ', $texto));
$unicas = array_unique($palabras, SORT_STRING);
$signos = array('.', ',', ';', ':', '"');
array_walk($unicas,
    function(&$w) use ($signos) { $w = str_replace($signos, '', $w); }
);
$keywords = array();
foreach($unicas as $palabra) if(strlen($palabra) > 2) $keywords[] = $palabra;
$string_keywords = join(", ", $keywords);print
```

Ejemplo de uso:

```
$texto = "Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan
euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo
dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit
definiebas an eos. Eu sit \"tincidunt\" incorrupte definitionem, vis mutat
affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine
equidem dolores: Quo no falli viris intellegam, ut fugit veritus placerat per.

Ius id vidit volumus mandamus, vide veritus democritum te nec, ei eos debet
libris consulatu. No mei ferri graeco dicunt, ad cum veri accommodare. Sed at
malis omnesque delicata, usu et iusto: zzril meliore. Dicunt maiorum eloquentiam
cum cu, sit summo dolor essent te. Ne quodsi nusquam legendos has, ea dicit
voluptua eloquentiam pro, ad sit quas qualisque. Eos vocibus deserunt quaestio
ei.";


// string de palabras clave separadas por coma
print strtolower($string_keywords);
/*
Salida:
lorem, ipsum, his, scripta, blandit, partiendo, eum, fastidii, accumsan,
euripidis, liber, hendrerit, qui, wisi, vocibus, suscipiantur, quo, dicit,
ridens, inciderint, quo, mundi, lobortis, reformidans, legimus, senserit,
definiebas, eos, sit, tincidunt, incorrupte, definitionem, vis, mutat, affert,
percipit, eirmod, consectetuer, signiferumque, per, usu, latine, equidem,
dolores, falli, viris, intellegam, fugit, veritus, placerat, ius, vidit,
volumus, mandamus, vide, democritum, nec, eos, debet, libris, consulatu, mei,
ferri, graeco, dicunt, cum, veri, accommodare, sed, malis, omnesque, delicata,
iusto, zzril, meliore, dicunt, maiorum, eloquentiam, summo, dolor, essent,
quodsi, nusquam, legendos, has, voluptua, pro, quas, qualisque, eos, deserunt,
quaestio
*/
```

#18:00 (tip) MVC

Tip para manejar un sistema ABM mediante URL amigables implementando un pseudo patrón MVC sin emplear OOP

Crea un archivo .php por cada tabla que debas administrar.

Por ejemplo, si tienes 3 tablas: categorias, productos y pedidos, tendrás 3 archivos: categorias.php, productos.php y pedidos.php

Asigna a cada archivo un *namespace* con el mismo nombre pero sin la extensión .php

Por ejemplo, el *namespace* del archivo productos.php será productos y el de pedidos.php será pedidos.

Unifica el mismo nombre de funciones en todos los archivos

Utiliza siempre los mismos nombres de funciones para cada acción (agregar, guardar, editar, actualizar, eliminar, ver, listar).

Así se vería un archivo llamado productos.php

```
namespace productos;

function agregar() {}     # muestra form para agregar
function guardar() {}     # hace un INSERT en la DB
function editar($id) {}   # muestra form para editar
function actualizar() {}  # hace un UPDATE en la DB
function listar() {}      # hace un SELECT en la DB de todos los registros
function eliminar($id) {} # hace un DELETE en la DB
function ver($id) {}      # hace un SELECT en la DB de un solo registro
```

Escribe una regla de reescritura por cada archivo que tengas, redirigiendo la solicitud a un único archivo llamado abm.php

Suponiendo los tres archivos de los ejemplos anteriores, tu .htaccess debería verse así:

```
RewriteEngine ON
RewriteRule ^categorias  abm.php
RewriteRule ^productos  abm.php
RewriteRule ^pedidos  abm.php
```


Crea un archivo llamado abm.php en la raíz de tu aplicación y *rutea* desde allí todas las solicitudes, de forma segura:

```
# Define un array SOLO con el nombre de los archivos que podrán llamarse
$archivos = array('productos', 'categorias', 'pedidos');

# Define otro array SOLO con el nombre de las funciones que podrán llamarse en cada archivo
$funciones = array('agregar', 'guardar', 'editar', 'actualizar', 'listar',
    'eliminar', 'ver');

$archivo = 'productos';  # define un archivo por defecto
$funcion = 'listar';     # define una función x defecto

# Las siguientes líneas se encargarán de ejecutar la función que corresponda
# a la solicitud del usuario
$uri = explode('/', $_SERVER['REQUEST_URI']);
if(@in_array($uri[1], $archivos)) $archivo = $uri[1];
if(@in_array($uri[2], $funciones)) $funcion = $uri[2];
$id = isset($uri[3]) ? (int)$uri[3] : 0;

require_once "{$archivo}.php";
$recurso = "{$archivo}\{$funcion}";
$recurso($id);
```

Ejemplo de accesos y *ruteos*:

```
http://www.example.org/productos/agregar   -> productos.php agregar()
http://www.example.org/productos/editar/15 -> productos.php editar(15)
http://www.example.org/productos/editar    -> productos.php editar(0)
http://www.example.org/categorias/ver/63   -> categorias.php ver(63)
```

Aviso: para que este *tip* funcione, debes tener habilitado el módulo rewrite en Apache. Si tienes acceso por consola al servidor, ejecuta el siguiente comando para habilitarlo:

```
sudo a2enmod rewrite && sudo service apache2 restart
```


#19:00 (hack) Vistas en MVC

Alternar entre menú de usuario y menú de administrador

Tener muchos archivos HTML separados es mejor que embeber código HTML en PHP pero sin embargo, existe una técnica mucho mejor aunque más avanzada, que es tener un único archivo HTML con todos los bloques necesarios. El hack para alternar las vistas, consiste en eliminar del HTML los bloques que no se necesitan dependiendo de determinadas condiciones. El mejor ejemplo es alternar la vista de un menú de administrador con uno de usuario común.

Dada una única plantilla HTML, se identifica el comienzo y final de los dos menús mediante comentarios HTML similares:

```
Archivo: tenplate.html
<!doctype html>
. . .
<body>
  <tags...>
  ......
  <!--menulogueado-->
    <nav>
        este es el menú del administrador
    </nav>
  <!--menulogueado-->

  <!--menuNOlogueado-->
    <nav>
        este es el menú para los usuarios comunes sin permisos
    </nav>
  <!--menuNOlogueado-->
.....
<otros>
<tags...>
```

Dependiendo del valor de una variable de sesión (es_admin), se establecerá el identificador del comentario HTML que diferencia ambos menús:

```
$id = isset($_SESSION['es_admin']) ? '' : 'NO';
```

Finalmente, se procede a eliminar el código que corresponda:

```
print preg_replace("/<!--menu{$id}logueado-->/", '', $html);
```


#20:00 (tip) Bases de datos

Ahorrar recursos reemplazando el uso de bases de datos por archivos

Muchas veces empleamos las denominadas «tablas codificadoras» para satisfacer requerimientos visuales. Por ejemplo, una lista de países en un formulario de contacto que solo será enviado por e-mail, no necesita de una tabla codificadora de países ya que estos datos no van a ser relacionados con otros.

Cuando las tablas codificadoras no se emplean en un contexto relacional (es decir, no enlazan ni son enlazadas con otras tablas) dejan de tener sentido por el gran consumo de recursos que generan las consultas y sus conexiones.

La alternativa «económica» es el uso de archivos de texto plano (NO CONFUNDIR con bases de datos NoSQL).

En un archivo de texto, se coloca una lista de los datos necesarios:

```
// archivo: paises.txt
Argentina
Mexico
Colombia
Venezuela
Chile
Ecuador
...
```

Luego se lo *parsea* en un array y se lo ordena de forma ascendente o descendente según se necesite:

```
# se utiliza sort para orden ascendente, rsort para descendente
$orden = "sort";
$paises = explode(chr(10), file_get_contents('paises.txt'));
$orden($paises);
print_r($paises);

// Array ( [0] => Argentina [1] => Chile [2] => Colombia [3] => Ecuador [4] =>
Mexico [5] => Venezuela )
```


#21:00 (trick & hack) Optimización de Bases de datos

Ordenar resultados de una consulta a base de datos sin usar SQL

Reducir costos a la hora de efectuar consultas a la base de datos, es fundamental en cualquier aplicación. Una forma de ahorrar recursos es utilizar funciones de ordenamiento de arrays en lugar de implementar la cláusula ORDER BY con campos no indexados.

Dada la siguiente consulta:

```
SELECT id, denominacion, precio FROM productos;
```

Y obtenidos los siguientes resultados:

```
$rows = array(
    array("id"=>1, "denominacion"=>"Pantalón largo", "precio"=>699.00),
    array("id"=>5, "denominacion"=>"Pollera", "precio"=>412.15),
    array("id"=>12, "denominacion"=>"Camisa manga corta", "precio"=>898.50),
    array("id"=>15, "denominacion"=>"Camisa manga larga", "precio"=>1200.00),
    array("id"=>16, "denominacion"=>"Remera sin mangas", "precio"=>250.00),
    array("id"=>21, "denominacion"=>"Musculosa", "precio"=>220.00),
    array("id"=>22, "denominacion"=>"Camiseta", "precio"=>300.00)
);
```

Emplear el siguiente **truco** para ordernarlos en **PHP 5.5**:

```
# Ordenar por el campo denominacion de forma ascendente (PHP 5.5)
array_multisort(array_column($rows, 'denominacion'), SORT_ASC, $rows);

# Ordenar por el campo precio de forma descendente (PHP 5.5)
array_multisort(array_column($rows, 'precio'), SORT_DESC, $rows);
```

Utilizar el siguiente **hack** para **PHP 5.3 y 5.4**:

```
foreach($resultados as $row) $productos[] = $row['denominacion'];
array_multisort($productos, SORT_ASC, $resultados);

foreach($resultados as $row) $precios[] = $row['denominacion'];
array_multisort($precios, SORT_DESC, $resultados);  # Descendente
```


#22:00 (truco) Optimización de Bases de datos

Filtrar resultados de una consulta a base de datos sin usar SQL

Otra forma de optimizar las consultas a bases de datos, es emplear una función de filtrado de arrays en lugar de utilizar la cláusula WHERE LIKE de SQL.

Dada la siguiente consulta (se evita el WHERE LIKE '%...%'):

```
SELECT id, denominacion, precio FROM productos;
```

Y obtenidos los siguientes resultados:

```
$rows = array(
    array("id"=>1, "denominacion"=>"Pantalón largo", "precio"=>699.00),
    array("id"=>5, "denominacion"=>"Pollera", "precio"=>412.15),
    array("id"=>12, "denominacion"=>"Camisa manga corta", "precio"=>898.50),
    array("id"=>15, "denominacion"=>"Camisa manga larga", "precio"=>1200.00),
    array("id"=>16, "denominacion"=>"Remera sin mangas", "precio"=>250.00),
    array("id"=>21, "denominacion"=>"Musculosa", "precio"=>220.00),
    array("id"=>22, "denominacion"=>"Camiseta", "precio"=>300.00)
);
```

Emplear el siguiente **truco** para filtrar los resultados:

```
function filter($row) {
    return (strpos($row['denominacion'], "manga") !== false);
}

$resultados = array_filter($rows, "filter");

// en negritas, el criterio de búsqueda
```


#23:00 (truco) Bases de datos

Generación dinámica de queries

Armar dinámicamente un *query* para seleccionar todos los registros de una tabla con solo contar con el nombre de la tabla es sencillo:

```
SELECT * FROM $tabla;
```

Sin embargo, no solo es una mala práctica, sino que además, no es muy útil si lo que se desea, es conocer el nombre de los campos y asociarlos a su valor correspondiente. Con este sencillo truco y la ayuda de MySQLiLayer[1], se pueden crear *queries* dinámicos incluyendo el nombre de los campos y obtener arrays asociativos de todos los registros:

```
function get_rows_from_table($tabla) {
    $sql = "SELECT COLUMN_NAME
            FROM   COLUMNS
            WHERE  TABLE_SCHEMA = ? AND TABLE_NAME = ?";
    $data = array('ss', "nombre_db", "$tabla");
    $fields = array('column'=>'');
    $rows = mysqli_run_query($sql, $data, $fields, 'INFORMATION_SCHEMA');
    $campos = array();
    // en PHP 5.5 se puede usar array_column
    foreach($rows as $row) $campos[$row['column']];
    $string_campos = join(', ', $campos);
    $nuevo_query = "SELECT $string_campos FROM ?";
    $data = array("s", "$tabla");
    return mysqli_run_query($nuevo_query, $data, $campos);
}
```

Llamando a `get_rows_from_table('productos')` se ejecutará un *query* como el siguiente:

```
SELECT id, denominacion, precio FROM productos;
```

Y se obtendrá un array de resultados como este:

```
$rows = array(
    array("id"=>1, "denominacion"=>"Pantalón largo", "precio"=>699.00),
    array("id"=>5, "denominacion"=>"Pollera", "precio"=>412.15),
    array("id"=>12, "denominacion"=>"Camisa manga corta", "precio"=>898.50),
    array("id"=>15, "denominacion"=>"Camisa manga larga", "precio"=>1200.00),
);
```

1 https://bazaar.launchpad.net/~eugeniabahit/europioexperimental/rel-3.5/view/head:/core/orm_engine/mysqlilayer.php

