Funciones de compresión zlib

Algunas de esas funciones son estas:

$f=**gzopen(***fich,mod, path***)**

Abre el fichero identificado por el parámetro *fich* y lo hace en modo especificado en el parámetro modo **r** o **w** según se trate de modo *lectura* o *escritura*.

Cuando se trata del modo de escritura el parámetro **w** debe ir seguido de un número comprendido entre **cero** y **nueve** que especifica el *grado de compresión* pretendido.

El parámetro *path* es opcional y puede contener un valor lógico (cero ó uno). Cuando el valor de este parámetro es **1** permite incluir en el parámetro *fich* **la ruta del directorio o subdirectorio** que alberga el fichero que tratamos de abrir

Si se incluye una *ruta* sin especificar el valor **1** en el parámetro *path* aparecerá un error.

gzclose(*$f***)**

Cierra el fichero asociado al identificador de recurso *$f*. Esta función devuelve TRUE en caso de éxito o FALSE si se produjera un error.

gzeof(*$f***)**

Esta función devuelve **1** (TRUE) en el caso de que *el puntero apunte al final del fichero* abierto e identificado mediante **$f**.
También devuelve TRUE en caso de error.
Si el fichero estuviera abierto y el puntero apunta a una posición distinta del final del fichero devolverá FALSE.

gzseek(*$f,desplaza***)**

Desplaza -dentro del fichero identificado por **$f**- el puntero -*a partir de su posición actual*- la cantidad de **bytes** indicados en el parámetro **desplaza**

gztell(*$f***)**

Devuelve la posición actual del puntero.

gzrewind(*$f***)**

Herramientas de compresión

Existen varias herramientas para compresión de ficheros. Las más populares son las funciones de la biblioteca **bzip2** de *Julian Seward* que generan ficheros comprimidos que se reconocen por su extensión (bz2) y la función de **zlib** de *Jean-loup Gailly y Mark Adler* para leer y grabar archivos comprimidos con extensión **gz**. En esta página veremos el uso de la segunda de las opciones. Empezaremos omprobando en *info.php* que la opción está activada. Deberemos ver algo como esto:

zlib

ZLib Support	enabled
Stream Wrapper support	compress.zlib://
Stream Filter support	zlib.inflate, zlib.deflate
Compiled Version	1.2.3
Linked Version	1.2.3

En la versión de PHP que estamos utilizando esta opción se activa por defecto y **no es necesario modificar ningún parámetro** en *php.ini*.

Ejemplo de compresión y lectura de un fichero

En este ejemplo trataremos de utilizar las funciones de compresión comentadas al margen. Si observas las formas de apertura de los ficheros verás que son similares a las utilizadas para la gestión de ficheros. Los modos de apertura para **escritura** son: **"w0"** a **"w9"** siendo los valores de **cero a nueve** los indicadores de los niveles de compresión. Para **lectura** debe usarse el modo **"r"** sin indicar ningún nivel de compresión.

```
<?
# asignamos un nombre al fichero con extensión "gz"
  $fichero ='prueba.gz';
# abrimos el fichero en modo escritura (w)
# con el nivel máximo de compresión (9)
  $f=gzopen($fichero,"w9",0);
  $cadena="Este es el primer bloque de texto que hemos
              introducido en el fichero comprimido. ";
  $cadena .="Añadimos este segundo bloque";
  echo "<i>Esta es la cadena inicial:</i> ".$cadena."<br>";
# escribimos (comprimida) la cadena en el fichero
  gzwrite($f,$cadena);
# cerramos el fichero
  gzclose($f);
#abrimos el fichero en modo lectura
  $f=gzopen($fichero,"r");
  echo "<i>Estos son los tres primeros caracteres de la cadena:</i> ";
# escribimos los tres primeros caracteres, el puntero (por defecto)
# apunta al comienzo de la cadena
 echo gzread($f, 3)."<br>";
# desplazamos el puntero hasta el carácter nº 8
  gzseek($f,8);
  echo "<i>Estos son los seis caracteres siguientes al octavo:</i> ";
# escribimos seis caracteres a partir del octavo
  echo gzread($f, 6)."<br>";
  echo "<i>Ahora el puntero está en:</i> ";
# buscamos la posición actual de puntero
  echo gztell($f)."<br>";
# movemos el puntero hasta el comienzo del fichero
    gzrewind($f);
echo "<i>Estos son los diez primeros caracteres de la cadena:</i> ";
```

Coloca el puntero al comienzo del fichero

gzread(*$f, longitud***)**

Devuelve una cadena -después de descomprimida- de longitud igual a la indicada en el parámetro **longitud**. La lectura comienza en la *posición actual del puntero* y acaba cuando la longitud de la cadena leida y descomprimida sea igual al valor del parámetro **longitud** o cuando se haya alcanzado el *final del fichero*.

gzpassthru (*$f***)**

Esta función **escribe** en la salida (no necesita la función echo) el contenido del fichero desde la posición actual del puntero hasta el final del fichero. Como es lógico, si estuviera *precedida* de **gzrewind** escribiría el fichero completo.

> **¡Cuidado!**
>
> La función **gzpassthru** cierra automáticamente el fichero después de escribir su contenido. Si pones **gzclose** después de esta función *te dará error* y si quieres seguir utilizando el fichero tendrás que volver a abrirlo con la función **gzopen**.

gzwrite(*$f, cadena, long***)**

Esta función **escribe** en el fichero comprimido que se identifica por **$f** la cadena contenida en el parámetro **cadena**.

Opcionalmente puede llevar el tercer parámetro (**longitud**) en cuyo caso solo escribirá los primeros *longitud* **bytes** de la cadena. Si el parámetro *longitud* existe y es *mayor* que la longitud de la cadena, insertará la *cadena completa*.

gzputs(*$f, cadena, long***)**

Esta función es idéntica a **gzwrite**.

readgzfile(*$fichero,path***)**

Esta función **abre de forma automática** el fichero indicado como parámetro **fichero**, además lo **lee** y lo **escribe de forma automática** sin necesidad de usar *echo* ni ninguna otra función de salida.

Si el fichero no está en el mismo directorio que el *script* -además de incluir la ruta en la cadena *fichero*- es necesario añadir el segundo parámetro -**path**- con valor **1**.

```
  echo gzread($f, 10)."<br>";
# volvemos el puntero al comienzo del fichero
    gzrewind($f);
  echo "<i>Escribimos el fichero completo:</i> ";
# con gzpasthru escribimos el fichero completo
# el puntero está al principio porque alli lo ha situado gzrewind
# no necesitamos utilizar "echo"  ni "print" ya que gzpassthru
# escribe directamente el contenido del fichero
  gzpassthru($f);
# tenemos que volver a abrir el fichero ya que gzpassthru
# se encargó de cerrarlo después de leerlo
 $f=gzopen($fichero,"r");
  echo "<br><i>Aquí estará todo el fichero:</i> ";
  gzpassthru ($f);
# la función readgzfile abre el fichero, imprime su contenido y lo cierra
  echo "<br><i>Aqui se imprime la cadena
                        completa usando readgzfile</i>: <br>";
  readgzfile($fichero);
/* con gzfile también se abre el fichero,
 pero ahora el contenido no se presenta
 directamente. Es recogido en un array.
 Para visualizarlo debemos imprimir
 el primer elemento del array. */
  $z=gzfile($fichero);
  echo "<br><i>Este es el primer elemento (0)
            del array generado por gzfile</i>: ".$z[0];
# gzfile cierra el fichero.
# No podemos poner gzclose porque nos daría error
 ?>
```

comprime1.php

Utilizando un directorio distinto

El ejemplo anterior está desarrollado para el supuesto que el *script* y el *fichero comprimido* estén en el mismo directorio.

Si quieres utilizar estas funciones utilizando ficheros alojados en un directorio distinto, solo tendrás que recordar que algunas funciones deben incluir el parámetro complementario **1**. Estos son las modificaciones que deberías efectuar:

La variable que recoge el nombre del fichero debe incluir el *path*, por ejemplo: **$fichero ='/subdirectorio/prueba.gz'**

La función **gzopen** debe incluir el tercer parámetro (path) con valor **1**, por ejemplo: **$f=gzopen($fichero,"r",1);**

También las funciones **gzfile** y **readgzfile** -que abren automáticamente el fichero- deberán incluir ese valor **1** como parámetro añadido. Por ejemplo: **readgzfile($fichero,1)** ó **$z=gzfile($fichero,1)**

Elección del grado óptimo de compresión

Puede parecer -a primera vista- que la condición óptima de compresión sería elegir el **nivel 9** y eso es cierto si tomamos únicamente en consideración el tamaño final del fichero comprimido. Sin embargo no existe una relación lineal entre reducción de tamaño/nivel de compresión. Sin que pueda considerarse ninguna referencia exacta -la compresión alcanzable depende del contenido del fichero y en consecuencia no puede establecerse una relación funcional puede comprobarse experimentalmente que -aparentemente- a partir del grado 2 la reducción de tamaño del fichero es mínima y que cuando se aumenta el grado de compresión a niveles máximos (tratándose de ficheros de un cierto tamaño) el tiempo de ejecución aumenta sustancialmente como consecuencia de la reiteración de la ejecución de los algoritmos de compresión.

Compresión de cadenas

En este ejemplo utilizamos *las tres funciones de compresión de cadenas* así como las opciones de descompresión y lectura de cada una de ellas.

```
<?
```

Comprimiendo cadenas

Las funciones anteriores permiten la creación, lectura y modificación de ficheros comprimidos

Sin embargo, existen otras funciones PHP que permiten comprimir *cadenas*. Aquí tienes algunas de ellas.

gzcompress(*cadena, nivel***)**

Esta función devuelve una **cadena comprimida** a partir de una *original* especificada en el parámet´ro **cadena**. El nivel de compresión (valores entre 0 y 9) se especifica en el parámetro **nivel**.

Las cadenas resultantes de esta función pueden *descomprimirse* aplicando la función **gzuncompress** que te comento más abajo.

gzdeflate(*cadena, nivel***)**

Se comporta de forma idéntica a la función anterior. La única salvedad parece ser que utiliza un *algoritmo de compresión distinto*.

Las cadenas resultantes de esta función también pueden *descomprimirse* aplicando la función **gzdeflate**.

gzencode(*cad, niv, opc***)**

Esta función devuelve la cadena **cad** comprimida con el nivel especificado en **niv** y permite dos opciones de compresión: **FORCE_GZIP** ó **FORCE_DEFLATE** que se pueden especificarse como tercer parámetro (**opc**) *sin encerrar entre comillas*.

El *valor por defecto* (cuando no se especifica el parámetro opción) es **FORCE_GZIP**

Descomprimiendo cadenas

gzuncompress(*cadena***)**

Con esta función se obtiene una cadena -descomprimida- *a partir de la cadena comprimida* indicada en el parámetro cadena- siempre que esta hubiera sido comprimida usando la función **gzcompress**

gzinflate(*cadena***)**

Funciona igual que la anterior. La única diferencia es que esta *descomprime* las cadenas que han sido comprimidas con **gzdeflate**

Funciones para *buferización* de salidas

ob_start()

Esta función activa la *buferización* de

```
# creamos una cadena de ejemplo
  $cadena="Esta es la cadena a comprimir. Intentaremos que sea larga
    porque parece que si la hacemos muy corta en vez de reducirse
    su tamaño parece que aumenta. Y como sigue siendo enormemente
    grande la cadena comprimida intentaremos hacerla aun mayor
    a ver que pasa ";
# comprimimos con la función gzcompress
  $c=gzcompress($cadena,9);
     echo "<br>".$c;
# descomprimimos con la función gzcompress
  $dc=gzuncompress($c);
     echo "<br>".$dc."<br>";
# ahora utilizamos la función gzencode
  $c1=gzencode($cadena,9,FORCE_GZIP);
  echo "<br>".$c1."<br>";
/* el resultado lo guardamos en un fichero con extensión gz
 pero abierto en modo "normal", es decir escribiendo
 dentro del fichero la cadena "tal cual" fue devuelta
 por gzencode*/
  $f=fopen("pepe.gz","w");
  fwrite($f,$c1);
  fclose($f);
# abrimos el fichero anterior utilizando las funciones
# de lectura de fichero comprimidos
  $f=gzopen("pepe.gz","r");
  readgzfile("pepe.gz");
  gzclose($f);
# borramos el fichero una vez leido
  unlink("pepe.gz");
# otra opción de compresión de cadenas utilizando la función
# gzdeflate
  $c2= gzdeflate($cadena,9);
    echo "<br><BR>".$c2;
# con la función gzinflate podemos descomprimir la cadena
# comprimida generada por gzdeflate
        $dc2=gzinflate($c2);
     echo "<br>".$dc2;
?>
```

comprime2.php

Economizando espacio en el servidor

Las opciones de compresión pueden permitirnos un cierto ahorro de espacio de servidor. Las páginas HTML podrían almacenarse comprimidas y ser *llamadas* a través de un script de descompresión que permita visualizarlas. En este ejemplo se efectúa la compresión de una página web (una de las páginas de estos materiales guardada en formato HTML) cuyo tamaño original es de 29.015 bytes. El fichero comprimido resultante ocupa 9.886 bytes. Como verás, el fichero se reduce a poco más del 30% del original.

PHP a partir de su activación.

Dicho de otra forma, *impide que las salidas generadas por el script se envíen al cliente y por tanto no serán visualizadas en el navegador*. A partir del momento de activar esa *buferización*, todas las salidas generadas se almacenan en una variable específica llamada: **ob_get_contents()**

ob_end_clean()

Esta función **desactiva** la *buferización* iniciada por **ob_start** y borra los contenidos de la variable **ob_get_contents()**

ob_clean()

Esta función *vacía* el buffer de salida pero sin *desactivar la bufferización*. Las salidas posteriores a esta función seguirían siendo recogidas en el *buffer*.

Cabeceras para transferir información comprimida

Cuando un servidor recibe una petición de una página web el navegador del cliente siempre envía información sobre su disposición a aceptar diferentes tipos de contenidos.

Una de las informaciones que suelen recibirse con la *petición del navegador* se refiere a su capacidad para aceptar *contenidos codificados* y esto suelen hacerlo mediante el envio de una cabecera que diría algo similar a esto: *Accept-Encoding:gzip,deflate* o *Accept-Encoding: gzip*.

Esta posibilidad es una característica común a todas las versiones modernas de los navegadores (es posible que algunas versiones antiguas no acepten esta codificación) y bastará con se incluya en la respuesta (el documento transferido por el servidor al cliente) la cabecera:
Header('Content-Encoding: gzip')
para que el navegador *sepa* que la *información llegará codificada* y que *debe activar* -de forma automática- *sus mecanismos de traducción de ese tipo de contenidos*.

Algunas limitaciones

En todos estos ejemplos hemos dado por supuesto que los navegadores de los clientes **aceptan la codificación gzip**, pero es evidente que si eso *no ocurriera* la página se visualizaría erróneamente.

Ejercicio nº 30

```
<?
# Creamos una variable "vacia"
$cadena="";
# Abrimos el fichero en modo lectura (r)
$f1=fopen("prueba.html","r");
/* hacemos un bucle para leer el fichero
  hasta encontrar el final (feof) y vamos recogiendo
  el contenido en la variable */
while (!feof($f1)) {
    $cadena .= fgets($f1, 1024);
  }
/*comprimimos la cadena con gzencode
  con lo cual la propia función añade los "encabezados"
  de formato gzip*/
$c1=gzencode($cadena,3,FORCE_GZIP);
/* abrimos un nuevo fichero modo escritura (w)
con "fopen", es decir como un fichero normal con extensión GZ */
 $f=fopen("prueba.html.gz","w");
/* escribimos la cadena "tal cual"
  en este fichero */
  fwrite($f,$c1);
# cerramos el fichero comprimido
  fclose($f);
  echo "La compresión ha terminado";
?>
```

comprime3.php

El fichero comprimido mediante el script anterior no puede ser visualizado directamente. Requiere ser descomprimido antes de ser enviado al navegador. Y eso podría hacerse mediante un script como este:

```
 <?
#abrimos el fichero comprimido con "gzopen"
 $f=gzopen("prueba.html.gz","r");
 /* leemos el contenido completo
  en forma transparente ya que readgzfile descomprime
  la salida*/
  readgzfile("prueba.html.gz");
  # cerramos el fichero
  gzclose($f);
?>
```

Visualizar fichero comprimido

Economizando tiempo de transferencia

No solo se puede economizar espacio en el servidor. También es posible enviar *comprimidas* -desde el servidor hasta el cliente- las páginas web.

En ese caso, será el propio navegador el que interprete la información comprimida y la presente de una manera transparente. Lo que habremos ahorrado habrá sido tiempo de transferencia pero, igual que ocurría en el comentario anterior, esa reducción del volumen de información a transferir afecta únicamente al contenido de la página y no a otros elementos que puede incluir, tales como imágenes, etcétera.

Este es un ejemplo de un script que **comprime** una página web y **la envía comprimida** al cliente.

```
<?
/* activamos la bufferización de la salida
 para que no se presenten los resultados del script
 directamente en la página
 ¡¡Cuidado con no dejar lineas en blanco delante del script
 ya que vamos a insertar luego Headers!! */
ob_start();
# abrimos y leemos el fichero html
$f1=fopen("prueba.html","r");
fpassthru($f1);
# recogemos el contenido del buffer  en la variable $cadena
$cadena = ob_get_contents();
```

Escribe un script que guarde en un fichero comprimido la imagen **aviones4.jpg** que tienes en el directorio *images*.

Crea un segundo script que permita visualizarla en el navegador y evalúa la economía de espacio que aporta la compresión de este tipo de archivos.

```
# comprimimos la cadea con gzencode
# para que incluya los encabezados "gzip"
 $cd=gzencode($cadena,3,FORCE_GZIP);
# desactivamos la "buferización"
 # y borramos el contenido del buffer
ob_end_clean();
# insertamos la cabeceras
 # indicando el tipo de contenido y el tamaño
        Header('Content-Encoding: gzip');
        Header('Content-Length: ' . strlen($cd));;
# presentamos el contenido (cadena comprimida) que será
# "traducido" automáticamente por el navegador
 echo $cd;
?>
```

Ejecutar script

El ejemplo anterior *comprimía* el contenido del fichero antes de enviarlo. En este que incluimos a continuación partimos del supuesto de que la página **ya está comprimida en el servidor**. Por tanto, tendremos que leer el fichero comprimido y enviarlo, de igual forma, al cliente.

```
<?
ob_start();
/* En este caso abrimos el fichero con "gzopen"
 ya que se trata de un fichero comprimido
# todo lo demás es idéntico al ejemplo anterior*/
$f1=gzopen("prueba.html.gz","r");
gzpassthru($f1);
$cadena = ob_get_contents();
 $cd=gzencode($cadena,3,FORCE_GZIP);
 ob_end_clean();
        Header('Content-Encoding: gzip');
        Header('Content-Length: ' . strlen($cd));
 echo $cd;
?>
```

Ejecutar script