

# La manipulation d'images avec PHP : librairie GD

par [Michaël](#)

Date de publication : 04/03/2006

Dernière mise à jour : 18/11/2006

Ce tutoriel vous permettra de créer et modifier vos images simplement avec PHP grâce à GD2.

- I - Introduction
  - I-1 - Téléchargements
  - I-2 - Introduction
  - I-3 - Installation
- II - On commence !
  - II-1 - Première image
  - II-2 - Zut, on a oublié quelque chose ! Les headers
  - II-3 - Correction du code :
  - II-4 - Enregistrer une image
- III - La puissance de GD
  - III-1 - Le dessin : rectangles, cercles, arcs, polygones, lignes
    - III-1.1 - Notions de base
    - III-1.2 - Les couleurs
    - III-1.3 - Les lignes
    - III-1.4 - Les rectangles
    - III-1.5 - Les cercles et les ellipses
    - III-1.6 - Les arcs de cercles
    - III-1.7 - Les polygones
  - III-2 - Des formes, rien que des formes ! On peut écrire aussi
  - III-3 - La manipulation d'images
    - III-3.1 - Fonctions utiles : ouvrir une image, connaître les dimensions
    - III-3.2 - Les effets
      - III-3.2.1 - Changer le gamma d'une image
      - III-3.2.2 - Copie et fusion d'images
      - III-3.2.3 - Redimensionner une image
- IV - Comment... ? (howto)
  - IV-1 - Mettre une légende sous une image
  - IV-2 - Mettre une couleur sur toute l'image
- V - Conclusion

## I - Introduction

### I-1 - Téléchargements

Pour consulter le tutoriel hors ligne, vous pouvez télécharger les fichiers suivants

Format	Liens	
Fichiers source archivés en tar.bz2 (341ko)	<a href="#">FTP</a>	<a href="#">HTTP</a>
Fichiers source archivés en tar.gz (345ko)	<a href="#">FTP</a>	<a href="#">HTTP</a>
Fichiers source archivés en zip (352ko)	<a href="#">FTP</a>	<a href="#">HTTP</a>

### I-2 - Introduction

GD est une librairie. C'est à dire que c'est un élément en plus que l'on va donner à PHP. Depuis PHP 4.3, une version de GD est incluse. Cela évite de devoir compiler plusieurs choses pour que tout fonctionne. La compilation de GD sera abordée ultérieurement étant donné que ce n'est pas très compliqué. Avant de commencer à travailler avec GD, il faut vérifier qu'il est déclaré dans la configuration de PHP. Ce tutoriel est réalisé avec PHP 4.4.0 et GD > 2.0. Pour savoir quelles versions vous possédez, créez ce fichier à la racine de votre serveur et lancez le. Si vous obtenez le message d'erreur, reportez vous à la section installation sinon, vous pouvez commencer.

```
<?php
echo "Vous avez PHP ".phpversion();
$gd_info = gd_info();
if(!$gd_info)
    die("<br />La librairie GD n'est pas installée !");

echo "<br />Vous avez GD {$gd_info['GD Version']}";
?>
```

### I-3 - Installation

Sous Linux, il faut éditer votre fichier php.ini. Généralement, il est dans /etc/php.ini ou /etc/php4/php.ini. Vous devez simplement ajouter la ligne suivante à la fin du fichier.

#### Ligne à ajouter dans le php.ini

```
extension=gd.so
# pour ajouter la ligne manuellement en console
echo extension=gd.so >> /etc/php.ini
```

Sous EasyPHP, c'est le même principe. Allez dans le dossier où est installé EasyPHP et entrez dans le dossier "apache". Editez le fichier php.ini en enlevant le ; devant l'instruction

```
extension=php_gd2.dll
```

```
;extension=php_exif.dll  
;extension=php_fdf.dll  
;extension=php_filepro.dll  
extension=php_gd2.dll ←  
;extension=php_gettext.dll  
;extension=php_hyperwave.dll  
;extension=php_iconv.dll
```



*Attention !*

*À partir de maintenant, GD est considéré comme activé. Si toutefois ça ne fonctionne pas, je vous invite à consulter les [forums de developpez.com](http://forums.developpez.com)*

## II - On commence !

### II-1 - Première image

Prenez votre éditeur de fichier favori et allons-y ! Si vous n'avez pas d'éditeur, voici un [lien intéressant](#) d'un comparatif des différents éditeurs. Créez le fichier 01\_first\_image.php Il existe plusieurs fonctions pour créer une image, cela dépend de ce que vous voulez faire. Nous allons en voir que deux. La première est imagecreate. Cette fonction crée une image à palette. Pour vous donner une idée, une image à palette est comme une image gif : elle est limitée à un maximum de 255 couleurs avec un canal alpha (transparence). C'est donc une image plus légère mais moins belle. Nous n'allons pas nous intéresser à cette fonction car elle produit des images pas très jolies. Nous allons donc voir imagecreatetruecolor qui crée une image en vraies couleurs (24 bits). Pour utiliser cette fonction, c'est simple.

#### 01\_first\_image.php

```
<?php
$x = 50; //largeur de mon image en PIXELS uniquement !
$y = 100; //hauteur de mon image en PIXELS uniquement !

/* on créé l'image en vraies couleurs avec une largeur de 50 pixels et une hauteur de 100 pixels */
$image = imagecreatetruecolor($x,$y);
imagepng($image); //renvoie une image sous format png
?>
```

Il est obligatoire d'attribuer le résultat de imagecreatetruecolor à une variable, sans quoi on ne pourra plus accéder à l'image. J'utilise ici la fonction imagepng que nous n'avons pas encore vue : elle sert à afficher une image au format png. Maintenant, appelez votre fichier dans votre navigateur.

### II-2 - Zut, on a oublié quelque chose ! Les headers

Mais que se passe-t-il ? Aucune image n'est affichée mais seulement un texte bizarre. C'est normal. C'est une erreur que l'on rencontre souvent quand on commence avec GD, je voulais vous la faire voir. Votre navigateur ne sait pas que le fichier qu'il doit afficher est une image au format png. Il faut lui dire en utilisant la fonction header de PHP. On a plusieurs headers (ou en-têtes) disponibles pour les images.

```
header("Content-type: image/png"); //format png
header("Content-type: image/jpeg"); //format jpeg
header("Content-type: image/gif"); //format gif
header("Content-type: image/psd"); //format psd *
header("Content-type: image/bmp"); //format bmp
header("Content-type: image/tiff"); //format tiff *
header("Content-type: image/jp2"); //format jp2 *
header("Content-type: image/iff"); //format iff *
header("Content-type: image/vnd.wap.wbmp"); //format wbmp *
header("Content-type: image/xbm");
header("Content-type: application/x-shockwave-flash"); //animation flash *
```

\* Les bibliothèques correspondantes doivent être installées.

### II-3 - Correction du code :

#### 02\_first\_image.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$x = 50; //largeur de mon image en PIXELS uniquement !
$y = 100; //hauteur de mon image en PIXELS uniquement !

/* on créé l'image en vraies couleurs avec une largeur de 50 pixels et une hauteur de 100 pixels */
```

## 02\_first\_image.php

```
$image = imagecreatetruecolor($x,$y);
imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

La fonction `imagedestroy` prend en argument la variable contenant l'image. Pour faire compliqué, on choisira `$image` comme nom de variable :). La fonction détruit l'image en mémoire, libérant ainsi de la place. Cette fonction est à exécuter en dernier dans le script sinon vous risquez d'avoir un petit carré avec une croix rouge sous IE ou "L'image `http://localhost/02_first_image.php` ne peut être affichée car elle contient des erreurs" si vous êtes sous Firefox. Utilisez `imagedestroy`, votre serveur ne pourra que mieux se porter.

En visualisant à nouveau le fichier dans votre navigateur, vous devriez voir un rectangle noir. En effet, GD attribue automatiquement la couleur noire sur toute l'image si rien ne dit le contraire (une fonction par exemple).

## II-4 - Enregistrer une image

On peut afficher une image en GD, c'est son rôle après tout mais on peut aussi enregistrer une image. Voici la liste des fonctions permettant d'afficher, enregistrer des images.

```
imagepng($image);
imagejpeg($image);
imagegd($image);
imagegd2($image);
imagegif($image);
imagewbmp($image);
imagexbm($image);
```

Toutes ces fonctions peuvent prendre un deuxième argument. C'est un chemin vers un fichier dans lequel sera enregistré l'image. Vous devez avoir les droits d'écriture dans le répertoire : GD ne sait pas changer les droits tout seul : ça deviendrait vite la pagaille si GD en était capable. Exemple d'utilisation :

```
<?php
$file = '/images/mon_image.png';
imagepng($image, $file); //enregistre l'image dans le répertoire images à la racine du site.
?>
```

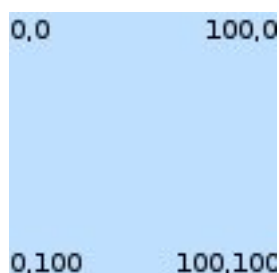
Heureusement, GD ne s'arrête pas là !

## III - La puissance de GD

### III-1 - Le dessin : rectangles, cercles, arcs, polygones, lignes

#### III-1.1 - Notions de base

GD sait dessiner des formes basiques si vous lui indiquez où est le crayon, le papier et ce que vous voulez dessiner. Tout d'abord, il faut connaître le système de coordonnées de GD. Pour GD, l'origine du carré est en haut à gauche. L'axe horizontal est l'axe des abscisses (x) et l'axe vertical est l'axe des ordonnées (y). Il n'y a pas de troisième dimension avec GD. On ne travaille qu'en 2D. Il y a des fois, c'est assez compliqué avec 2 dimensions seulement.



#### III-1.2 - Les couleurs

Commençons par créer une couleur. La fonction `imagecolorallocate` a deux types de fonctionnements. Elle n'accepte que des valeurs en RVB soit en hexadécimal soit des entiers compris entre 0 et 255. Créons le bleu utilisé ci-dessus. Son code hexadécimal est `#BEDFFE`. Voici la séparation en trois canaux :

- Rouge : 0xBE
- Vert : 0xDF
- Bleu : 0xFE

Le code de conversion est inclus dans l'exemple :)

Voici le code permettant de créer la couleur :

##### 03\_color\_create.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$x = 50; //largeur de mon image en PIXELS uniquement !
$y = 100; //hauteur de mon image en PIXELS uniquement !

/* on créé l'image en vraies couleurs avec une largeur de 50 pixels et une hauteur de 100 pixels */
$image = imagecreatetruecolor($x,$y);
$color = "BEDFFE";
$rouge = hexdec(substr($color,0,2)); //conversion du canal rouge
$vert = hexdec(substr($color,2,4)); //conversion du canal vert
$bleu = hexdec(substr($color,4,6)); //conversion du canal bleu

$couleur = imagecolorallocate($image,$rouge,$vert,$bleu);
/* on créé la couleur et on l'attribue à une variable pour ne pas la perdre */

imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

Le premier argument de `imagecolorallocate` est l'image qui va disposer de la nouvelle couleur. Ensuite, ce sont les valeurs de chaque canal de couleur. Dans ce cas, elles sont envoyées sous forme d'entiers grâce à la fonction `hexdec`.

Profitions-en pour voir les couleurs transparentes. C'est la fonction `imagecolorallocatealpha` qui va nous le permettre.

```
$couleur_alpha = imagecolorallocatealpha($image, $rouge, $vert, $bleu, $alpha);
```

Cette fonction fonctionne comme `imagecolorallocate` à une différence près : l'argument `$alpha` supplémentaire. `$alpha` prend une valeur entière comprise entre 0 (opaque) et 127 (transparence).

#### 04\_imagecolorallocatealpha.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !

/* on créé l'image en vraies couleurs avec une largeur de 200 pixels et une hauteur de 200 pixels */
$image = imagecreatefromjpeg($_SERVER['DOCUMENT_ROOT']."/tuto_gd/background.jpg");

$color = "000000";
$rouge = hexdec(substr($color,0,2)); //conversion du canal rouge
$vert = hexdec(substr($color,2,4)); //conversion du canal vert
$bleu = hexdec(substr($color,4,6)); //conversion du canal bleu

/* on créé la couleur et on l'attribue à une variable pour ne pas la perdre */
$couleur = imagecolorallocatealpha($image,$rouge,$vert,$bleu,0);

putenv('GDFONTPATH=' . realpath('.')); //fonction inconnue

for($i=1;$i<4;$i++)
{
    $couleur = imagecolorallocatealpha($image,$rouge,$vert,$bleu,30*$i);
    imagettftext($image, 21, 0, 5, 50*$i+15, $couleur, 'hardway', 'transparence'); //fonction
    inconnue
}

imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

Dans cet exemple, deux fonctions encore inconnues sont utilisées (`putenv` et `imagettftext`). Nous les verrons plus loin dans ce tutoriel.

Maintenant que l'on sait déclarer une couleur, voyons comment appliquer une couleur de fond à une image.

#### 05\_image\_fill.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$x = 50; //largeur de mon image en PIXELS uniquement !
$y = 100; //hauteur de mon image en PIXELS uniquement !

/* on créé l'image en vraies couleurs avec une largeur de 50 pixels et une hauteur de 100 pixels */
$image = imagecreatetruecolor($x,$y);

$color = "BEDFFE";
$rouge = hexdec(substr($color,0,2)); //conversion du canal rouge
$vert = hexdec(substr($color,2,4)); //conversion du canal vert
$bleu = hexdec(substr($color,4,6)); //conversion du canal bleu

/* on créé la couleur et on l'attribue à une variable pour ne pas la perdre */
$couleur = imagecolorallocate($image,$rouge,$vert,$bleu);
imagefill($image,0,0,$couleur); //on remplit l'image
imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire ?>
```

La fonction utilisée ici est `imagefill`.

```
imagefill($image,$x,$y,$couleur);
```

Le premier argument est l'image à remplir, le deuxième et troisième représentent le point d'origine du remplissage et le quatrième est la couleur de remplissage. Cette fonction part d'un point ( $x,y$ ) et remplit toute la zone située en dessous de ce point. Dans notre exemple, on prend 0,0 comme origine pour remplir toute l'image. Vous devriez voir un rectangle bleu clair.

On continue avec quelque chose de simple : dessiner une ligne.

### III-1.3 - Les lignes

```
imageline($image, $x_orig, $y_orig, $x_dest, $y_dest, $couleur);
```

Le premier argument est l'image sur laquelle on va employer nos talents de dessinateurs. Le couple  $x\_orig$  et  $y\_orig$  désignent respectivement l'abscisse et l'ordonnée du point d'origine de la ligne.  $x\_dest$  et  $y\_dest$  désignent les coordonnées du point de fin de la ligne.  $couleur$  désigne bien entendu la couleur de la ligne.

#### 06\_create\_line.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$x = 50; //largeur de mon image en PIXELS uniquement !
$y = 100; //hauteur de mon image en PIXELS uniquement !

/* on créé l'image en vraies couleurs avec une largeur de 50 pixels et une hauteur de 100 pixels */
$image = imagecreatetruecolor($x,$y);

$color = "BEDFFE";
$rouge = hexdec(substr($color,0,2)); //conversion du canal rouge
$vert = hexdec(substr($color,2,4)); //conversion du canal vert
$bleu = hexdec(substr($color,4,6)); //conversion du canal bleu

/* on créé la couleur et on l'attribue à une variable pour ne pas la perdre */
$couleur = imagecolorallocate($image,$rouge,$vert,$bleu);

imageline($image,10,10,45,90,$couleur); //on créé une ligne
imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

Vous devriez voir ceci



Pour changer la taille de la ligne, il faut utiliser `imagesetthickness`

```
imagesetthickness($image,$size);
```

`$image` est l'image qui reçoit la modification. `$size` est l'épaisseur de la ligne en pixels. Vous pouvez également dessiner une ligne en pointillés grâce à `imagedashedline`. Son utilisation est similaire à `imageline`.

### III-1.4 - Les rectangles

Maintenant dessinons un rectangle.

Il existe encore une fois plusieurs fonctions pour dessiner un rectangle. Vous ne le saviez pas, il existe différents types de rectangles. Mais si ! Les rectangles vides et ceux qui sont remplis. La première fonction est `imagerectangle`. On l'utilise comme ceci

```
imagerectangle($image,$x_orig,$y_orig,$x_dest,$y_dest,$couleur);
```

Pour changer, le premier argument est l'image de destination. Ensuite, on doit indiquer un couple de coordonnées (`$x_orig`,`$y_orig`) qui définit le point de départ pour dessiner le rectangle. On doit aussi indiquer un deuxième couple de coordonnées (`$x_dest`,`$y_dest`) pour repérer le point de fin de dessin.

Chose importante : GD dessine à partir d'en haut à gauche. Si les coordonnées du point de fin ont une valeur inférieure au point de départ, vous risquez d'avoir des surprises.

#### 07\_create\_rectangle.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$x = 50; //largeur de mon image en PIXELS uniquement !
$y = 100; //hauteur de mon image en PIXELS uniquement !
/* on créé l'image en vraies couleurs avec une largeur de 50 pixels et une hauteur de 100 pixels */
$image = imagecreatetruecolor($x,$y);

$color = "BEDFFE";
$rouge = hexdec(substr($color,0,2)); //conversion du canal rouge
$vert = hexdec(substr($color,2,4)); //conversion du canal vert
$bleu = hexdec(substr($color,4,6)); //conversion du canal bleu

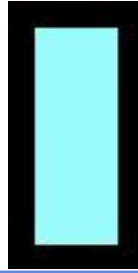
/* on créé la couleur et on l'attribue à une variable pour ne pas la perdre */
$couleur = imagecolorallocate($image,$rouge,$vert,$bleu);

imagerectangle($image,10,10,40,90,$couleur);
imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

Vous devriez avoir ceci avec `imagerectangle`



`imagefilledrectangle` fonctionne de la même manière que `imagerectangle`. En changeant le nom de la fonction, vous devriez avoir ceci



### III-1.5 - Les cercles et les ellipses

Vous savez tout sur les rectangles. Maintenant les cercles. Comme vous le savez certainement, un cercle est une ellipse dont la petite hauteur est égale à la grande hauteur. Ça tombe plutôt bien puisqu'il n'existe pas de fonction pour créer des cercles. C'est la fonction `imageellipse` qui permet de dessiner une ellipse ou un cercle.

```
imageellipse($image,$centre_x, $centre_y, $largeur, $hauteur, $couleur);
```

Pour créer un cercle, la hauteur et la largeur devront être égales, c'est simple non ?

#### 09\_create\_circle\_ellipse.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$x = 50; //largeur de mon image en PIXELS uniquement !
$y = 100; //hauteur de mon image en PIXELS uniquement !

/* on créé l'image en vraies couleurs avec une largeur de 50 pixels et une hauteur de 100 pixels */
$image = imagecreatetruecolor($x,$y);

$color = "BEDFFE";
$rouge = hexdec(substr($color,0,2)); //conversion du canal rouge
$vert = hexdec(substr($color,2,4)); //conversion du canal vert
$bleu = hexdec(substr($color,4,6)); //conversion du canal bleu
$couleur = imagecolorallocate($image,$rouge,$vert,$bleu);

/* on créé la couleur et on l'attribue à une variable pour ne pas la perdre */
imageellipse($image,25,50,25,25,$couleur); //on créé un cercle
imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

Vous devriez voir ceci avec `imageellipse`



`imagefilledellipse` fonctionne de la même manière que `imageellipse`. En changeant le nom de la fonction, vous devriez avoir ceci



### III-1.6 - Les arcs de cercles

Maintenant, les arcs. C'est `imagearc` que l'on va utiliser :

```
imagearc($image, $abscisse, $ordonnee, $largeur, $hauteur, $angle_debut, $angle_fin, $couleur);
```

La seule chose à laquelle on doit faire attention, c'est que GD commence son arc à gauche et évolue dans le sens contraire des aiguilles d'une montre. Les angles devront être exprimés en degrés : je vois les ennemis des radians soulagés



#### 11\_create\_arc.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$x = 50; //largeur de mon image en PIXELS uniquement !
$y = 100; //hauteur de mon image en PIXELS uniquement !
/* on créé l'image en vraies couleurs avec une largeur de 50 pixels et une hauteur de 100 pixels */
$image = imagecreatetruecolor($x,$y);

$color = "BEDFFE";
$rouge = hexdec(substr($color,0,2)); //conversion du canal rouge
$vert = hexdec(substr($color,2,4)); //conversion du canal vert
$bleu = hexdec(substr($color,4,6)); //conversion du canal bleu
/* on créé la couleur et on l'attribue à une variable pour ne pas la perdre */
$couleur = imagecolorallocate($image,$rouge,$vert,$bleu);

imagearc($image,25,50,20,20,0,180,$couleur); //on créé un arc avec une ouverture de 180 degrés
imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

Vous devriez voir ceci



Pour remplir l'arc, on utilisera bien sûr la fonction `imagefilledarc` mais elle demande un argument supplémentaire. L'argument à mettre est limité à 4 choix.

- 1 IMG\_ARC\_PIE : permet de créer une courbe entre deux points et de remplir
- 2 IMG\_ARC\_CHORD : permet de créer une ligne droite entre deux points et de remplir
- 3 IMG\_ARC\_NOFILL : permet de créer une ligne droite entre deux points et de **NE PAS** remplir
- 4 IMG\_ARC\_EDGED : permet de créer une ligne droite entre un point de l'arc et son centre : utile avec IMG\_ARC\_NOFILL pour créer des camemberts. Pour remplir l'arc, on utilisera bien sûr la fonction imagefilledarc mais elle demande un argument supplémentaire. L'argument à mettre est limité à 4 choix.

En remplaçant la ligne

```
imagearc($image,25,50,20,20,0,180,$couleur); //on créé un arc avec une ouverture de 180 degrés
```

par

```
imagefilledarc($image,25,50,20,20,0,180,$couleur,INT_ARC_PIE); //on créé un arc rempli avec une ouverture de 180 degrés
```

Vous devriez obtenir ceci



### III-1.7 - Les polygones

Maintenant, on attaque la fonction de dessin la plus complexe de GD : la création des polygones.

```
imagepolygon($image, $tableau_points, $nombre_points, $couleur);
```

Les premier et dernier arguments sont maintenant banals ... Le deuxième argument est un tableau qui contient les coordonnées des points limitants le polygone. Le troisième argument est le nombre de points qui constituent le polygone. À première vue, c'est un peu bête de lui indiquer le nombre de points puisqu'il a les coordonnées. Oui mais ça permet à la fonction de vérifier que les arguments sont corrects et qu'elle va pouvoir dessiner. Alors dessinons un triangle.

#### 13\_create\_polygon.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$x = 200; //largeur de mon image en PIXELS uniquement !
$y = 200; //hauteur de mon image en PIXELS uniquement !

/* on créé l'image en vraies couleurs avec une largeur de 200 pixels et une hauteur de 200 pixels */
$image = imagecreatetruecolor($x,$y);

$color = "BEDFFE";
$rouge = hexdec(substr($color,0,2)); //conversion du canal rouge
$vert = hexdec(substr($color,2,4)); //conversion du canal vert
$bleu = hexdec(substr($color,4,6)); //conversion du canal bleu

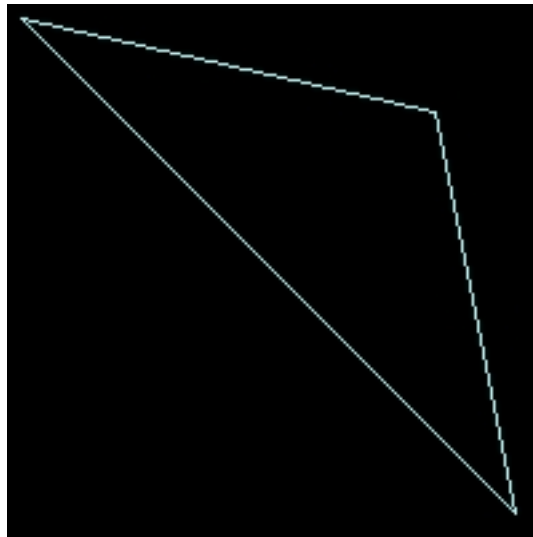
/* on créé la couleur et on l'attribue à une variable pour ne pas la perdre */
$couleur = imagecolorallocate($image,$rouge,$vert,$bleu);
```

## 13\_create\_polygon.php

```
/* tableau contenant les coordonnées des points toujours en couple (x,y) */
$coords = array('5','5',      // point 1
               '190','190',  // point 2
               '160','40');  // point 3

imagepolygon($image,$coords,3,$couleur); //on créé un polygone avec 3 points
imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

Vous devriez obtenir ceci avec imagepolygon



Vous devriez obtenir ceci avec imagefilledpolygon



### III-2 - Des formes, rien que des formes ! On peut écrire aussi

GD ne sait pas que dessiner. Il sait aussi écrire. On peut écrire horizontalement et verticalement. La première fonction est pour écrire horizontalement. En utilisant directement cette fonction, nous n'avons que 5 tailles d'une

même police disponible.

```
imagestring($image, $font, $x, $y, $string, $couleur);
```

\$font est un entier compris entre 1 et 5. 1 correspond à la plus petite taille tandis que 5 correspond à la plus grande. \$x et \$y donnent les coordonnées du point de départ. \$string est la chaîne de caractères à écrire. Attention, GD ne sait pas renvoyer automatiquement à la ligne si la chaîne est trop longue. Il faudra créer une fonction pour calculer quand renvoyer à la ligne.

#### 15\_write\_text

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$x = 200; //largeur de mon image en PIXELS uniquement !
$y = 200; //hauteur de mon image en PIXELS uniquement !

/* on créé l'image en vraies couleurs avec une largeur de 200 pixels et une hauteur de 200 pixels */
$image = imagecreatetruecolor($x,$y);

$color = "BEDFFE";
$rouge = hexdec(substr($color,0,2)); //conversion du canal rouge
$vert = hexdec(substr($color,2,4)); //conversion du canal vert
$bleu = hexdec(substr($color,4,6)); //conversion du canal bleu

/* on créé la couleur et on l'attribue à une variable pour ne pas la perdre */
$couleur = imagecolorallocate($image,$rouge,$vert,$bleu);

imagestring($image, 3, 65, 10, "horizontal", $couleur); //on écrit horizontalement
imagestringup($image, 3, 5, 180, "vertical", $couleur); //on écrit verticalement
imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

Vous devriez obtenir ceci



Mais ce n'est pas très joli comme police de caractères. Heureusement il est possible d'avoir de meilleures polices. Il y a une méthode un peu compliquée : il faut avoir des polices au format gdf. Ce format de police n'est pas le plus courant. Nous allons tout de même voir la fonction qui permet de gérer ces polices.

```
$font = imageloadfont($fichier);
```

On doit attribuer le résultat de `imageloadfont` à une variable pour ne pas perdre la police chargée. `$fichier` est le chemin vers le nom du fichier `.gdf` contenant la police.

#### 16\_loadfont.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$x = 200; //largeur de mon image en PIXELS uniquement !
$y = 200; //hauteur de mon image en PIXELS uniquement !

/* on créé l'image en vraies couleurs avec une largeur de 200 pixels et une hauteur de 200 pixels */
$image = imagecreatetruecolor($x,$y);

$color = "BEDFFE";
$rouge = hexdec(substr($color,0,2)); //conversion du canal rouge
$vert = hexdec(substr($color,2,4)); //conversion du canal vert
$bleu = hexdec(substr($color,4,6)); //conversion du canal bleu

/* on créé la couleur et on l'attribue à une variable pour ne pas la perdre */
$couleur = imagecolorallocate($image,$rouge,$vert,$bleu);

$font = imageloadfont('font.gdf'); //on charge la police

imagestring($image, $font, 5, 10, "horizontal", $couleur); //on écrit horizontalement
imagestringup($image, 3, 5, 180, "vertical", $couleur); //on écrit verticalement
imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

Vous devriez voir ceci



On voit bien ici que GD ne sait pas renvoyer à la ligne. La fonction `imageloadfont` est limitante tant par le format exigé que par l'unique taille des polices. On ne pourra pas avoir plusieurs tailles avec un seul fichier. C'est pour cela que la fonction `imagettftext` existe.

#### **ATTENTION !**

***Cette fonction nécessite que GD soit compilé avec le support de FreeType 2.x***

On peut écrire avec des polices au format ttf qui est un des formats les plus répandus. Vous n'aurez pas de problèmes à en trouver.

```
$points = imagettftext($image, $size, $angle, $x, $y, $couleur, $font_filename, $text);
```

Cette fonction renvoie un tableau de 8 éléments. Cela représente 4 couples de coordonnées des points remarquable du texte. Cela permet notamment de dessiner un rectangle autour du texte. Dans l'ordre, on a inférieur gauche, inférieur droit, supérieur droit, supérieur gauche. \$size est la taille en pixels (GD2) ou en points (GD1) de la police. \$angle est l'angle en degrés que le texte fera avec l'horizontale : 0° pour que le texte soit horizontal. \$font\_filename est le nom de fichier de la police TTF à utiliser. On peut mettre ou ne pas mettre l'extension, GD saura se débrouiller seul. Ai-je besoin de dire le rôle de \$text ?

### 17\_imagettftext.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$x = 200; //largeur de mon image en PIXELS uniquement !
$y = 200; //hauteur de mon image en PIXELS uniquement !

/* on créé l'image en vraies couleurs avec une largeur de 200 pixels et une hauteur de 200 pixels */
$image = imagecreatetruecolor($x,$y);

$color = "BEDFFE";
$rouge = hexdec(substr($color,0,2)); //conversion du canal rouge
$vert = hexdec(substr($color,2,2)); //conversion du canal vert
$bleu = hexdec(substr($color,4,2)); //conversion du canal bleu

/* on créé la couleur et on l'attribue à une variable pour ne pas la perdre */
$couleur = imagecolorallocate($image,$rouge,$vert,$bleu);

putenv('GDFONTPATH=' . realpath('.')); //ligne obligatoire !
imagettftext($image, 14, 45, 10, 190, $couleur, 'dark', 'Voici un texte !');

imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

La fonction putenv définit une variable d'environnement. Elle est obligatoire ! Si vous ne la mettez pas, vous ne définissez pas l'endroit où est stocké la police et cela renverra une erreur. Si vous avez la possibilité d'accéder à cette fonction, vous devriez obtenir ceci



## III-3 - La manipulation d'images

### III-3.1 - Fonctions utiles : ouvrir une image, connaître les dimensions

GD sait dessiner. C'est bien mais GD sait aussi manipuler les images. C'est utile si on veut créer un album photo avec des vignettes par exemple sans prendre chaque photo et les réduire manuellement pour créer les vignettes. Ça va bien quand il y a à peine 10 photos mais quand il y en a beaucoup plus (merci le numérique), on aime bien les fonctions de GD

Pour manipuler une image, il faut d'abord l'ouvrir. Dans ce cas, il faut connaître l'extension du fichier. On travaillera uniquement avec des images jpeg pour simplifier. On doit utiliser la fonction `imagecreatefromjpeg` pour ouvrir une image. Elle ne reçoit qu'un unique argument : le chemin de l'image avec son nom. Comme on crée une image, il faut attribuer le résultat de la fonction à une variable.

```
$image = imagecreatefromjpeg($file);
```

Dans ce tutoriel, les images sont situées à la racine du serveur web dans le répertoire nommé `images`. Vous aurez des erreurs si vous ne modifiez pas le chemin dans le fichier php et que les images ne sont pas à l'endroit voulu.

#### 18\_open\_image.php

```
<?
header('Content-type: image/png');
/* ligne à modifier selon votre installation */
$file = $_SERVER['DOCUMENT_ROOT'].'/images/first_open.jpg';
$image = imagecreatefromjpeg($file); //ouverture de l'image jpeg

imagepng($image);
imagedestroy($image);
?>
```

On voit dans cet exemple que GD est assez souple pour les types de fichier. Je charge une image jpg et je la renvoie sous format png. En ouvrant le fichier avec la fonction GD, il est automatiquement converti au format GD en mémoire permettant ainsi de pouvoir enregistrer dans le format qu'on veut. GD peut alors servir de convertisseur d'images.

Voici la liste des fonctions permettant d'ouvrir des images

```
imagecreatefromgd2($file); //ouvre un fichier gd2
imagecreatefromgd2part($file); //ouvre une partie d'un fichier gd2
imagecreatefromgd($file); //ouvre un fichier gd
imagecreatefromgif($file); //ouvre un fichier gif
imagecreatefromjpeg($file); //ouvre un fichier jpeg
imagecreatefrompng($file); //ouvre un fichier png
imagecreatefromwbmp($file); //ouvre un fichier wbmp
imagecreatefromxbm($file); //ouvre un fichier xbm
imagecreatefromxpm($file); //ouvre un fichier xpm
```

Il est souvent utile de connaître les dimensions d'une image. Par exemple, c'est utile si on veut limiter la taille d'une image uploadée. Il existe trois fonctions permettant de connaître les dimensions. `getimagesize` permet de connaître les dimensions et d'autres détails sur l'image en renvoyant un tableau.

```
$details = getimagesize($file);
```

Voici le contenu de `$details` :

- `$details[0]` -> largeur
- `$details[1]` -> hauteur
- `$details[2]` -> type d'image
- `$details[3]` -> ligne à mettre dans la balise `<img>` (`height=x width=y`)

Le type d'image cité ci-dessus est un chiffre. Voici la liste

- 1 -> GIF
- 2 -> JPG
- 3 -> PNG
- 4 -> SWF
- 5 -> PSD
- 6 -> BMP
- 7 -> TIFF (INTEL®)
- 8 -> TIFF (MOTOROLA®)
- 9 -> JPC
- 10 -> JP2
- 11 -> JPX
- 12 -> JB2
- 13 -> SWC
- 14 -> IFF

Cette fonction fait partie de PHP : il n'y a pas besoin de GD pour l'utiliser mais il est important de signaler son existence ici. Il existe des fonctions moins poussées avec GD. Avec GD, on doit obligatoirement ouvrir l'image et l'assigner à une variable alors qu'avec `getimagesize`, on n'a pas besoin de l'attribuer à une variable. Avec GD, on ne peut obtenir que les dimensions grâce à `imagesx` et `imagesy`.

```
$x = imagesx($image);
$y = imagesy($image);
```

### 19\_getimagesxy.php

```
<?
/* on récupère les infos sur l'image grâce à la fonction PHP */
$details = getimagesize($_SERVER['DOCUMENT_ROOT'].'/images/first_open.jpg');

/* on affiche les infos */
echo "PHP : L'image a une largeur de {$details[0]}px et une hauteur de {$details[1]}px. Elle est de
type ".image_type_to_mime_type($details[2]);

/* avec GD, on est obligé de créer l'image avant d'en faire quoi que ce soit */
$image = imagecreatefromjpeg($_SERVER['DOCUMENT_ROOT'].'/images/first_open.jpg');
$x = imagesx($image); //on récupère la largeur
$y = imagesy($image); //on récupère la hauteur

imagedestroy($image);
echo "<br />GD : L'image a une largeur de {$x}px et une hauteur de {$y}px";
?>
```

La fonction `image_type_to_mime_type` permet de connaître le format du fichier grâce au troisième champ du tableau qu'a renvoyé la fonction `getimagesize`. On voit d'après cette exemple qu'il est plus simple d'utiliser `getimagesize` que les fonctions GD. Avec GD, on doit faire plus de manipulations alors qu'une seule fonction suffit en PHP. Malgré ses qualités, GD n'est pas parfait ;) Il est donc conseillé d'utiliser `getimagesize`. Pour encore en savoir plus sur les images, il faut utiliser l'extension EXIF : ce tutoriel n'en traite pas.

## III-3.2 - Les effets

### III-3.2.1 - Changer le gamma d'une image

Avec GD, on peut faire bien plus que connaître les dimensions des images ou dessiner quelques formes basiques. On peut aussi travailler avec la transparence, la modification de taille, la découpe, etc. Alors commençons par éclaircir une image. Cela vous est sans doute arrivé : prendre une photo et que cette dernière soit trop sombre. On peut l'éclaircir avec GD grâce à la fonction `imagegammacorrect`.

```
imagegammacorrect($image, $gamma_in, $gamma_out);
```

`$gamma_in` est le seuil de gamma original. Par défaut, mettez le à 1.

`$gamma_out` est le seuil de gamma en sortie. Si par exemple vous mettez 2, la valeur du seuil aura doublé et votre image s'éclaircira. Il est également possible d'assombrir l'image en mettant une valeur de sortie inférieure à la valeur originale.

#### 20\_imagegammacorrect.php

```
<?php
header("Content-type: image/png"); //la ligne qui change tout !
$image = imagecreatefromjpeg("dark_image.jpg");

imagegammacorrect($image,1,2.2);

imagepng($image); //renvoie une image sous format png
imagedestroy($image); //détruit l'image, libérant ainsi de la mémoire
?>
```

Photo originale



Photo modifiée

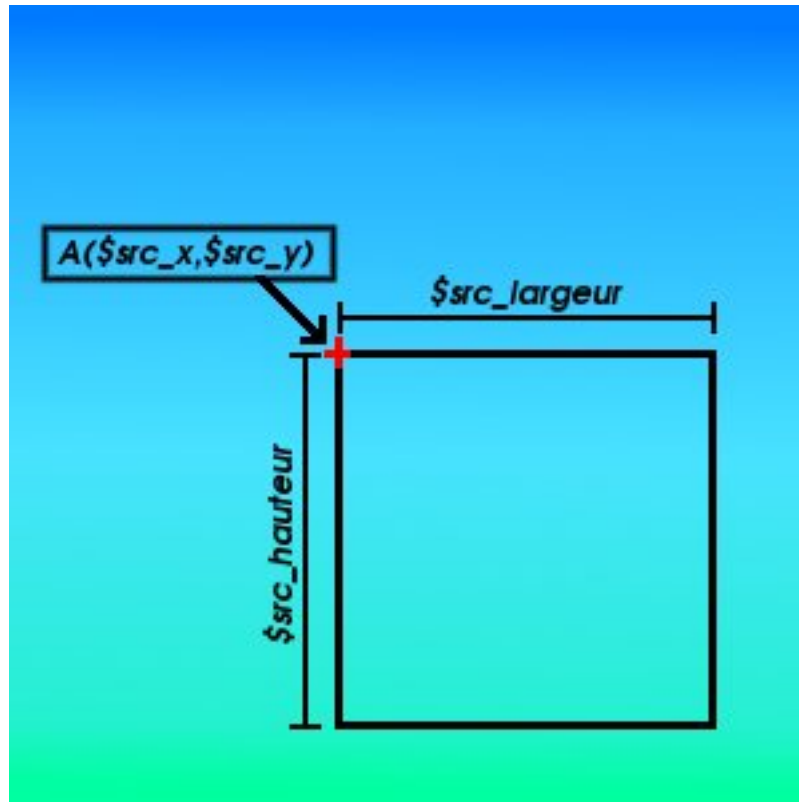


### III-3.2.2 - Copie et fusion d'images

Nous allons maintenant voir comment ajouter une image sur une autre image :) C'est utile pour mettre votre logo pour dire qu'une image vous appartient. Il existe plusieurs fonctions pour copier une image sur une autre. Chaque fonction a sa spécialité. La première copie simplement l'image.

```
imagecopy($image_dest, $image_src, $dest_x, $dest_y, $src_x, $src_y, $src_largeur, $src_hauteur);
```

`$image_dest` est l'image de destination. `$image_src` est l'image source (celle qui va être copiée). `$dest_x` et `$dest_y` sont les coordonnées du point de départ sur l'image de destination. `$src_x` et `$src_y` sont les coordonnées du point de départ de copie sur l'image source. `$src_largeur` est la largeur en pixels à recopier. `$src_hauteur` est la hauteur en pixels à recopier.



Le contenu du carré noir sera recopié vers l'image de destination.

#### 21\_imagecopy.php

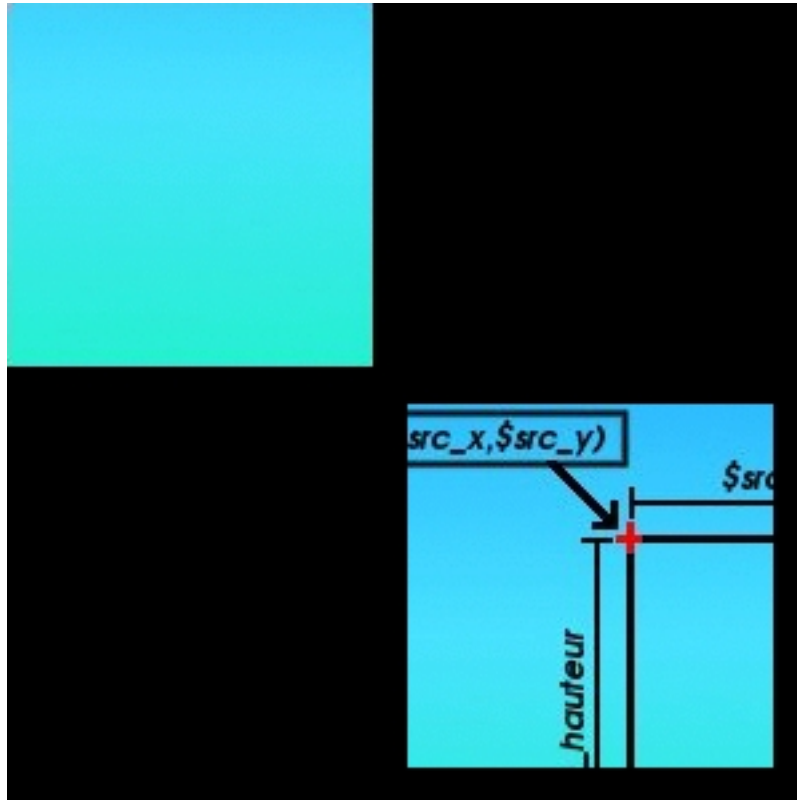
```
<?
header("Content-type: image/png"); //on envoie les infos au navigateur

$source = imagecreatefromjpeg("imagecopy.jpg"); //on ouvre l'image source
$destination = imagecreatetruecolor(300,300); //on créé une image truecolor de 300x300 pixels

imagecopy($destination,$source, 0, 0, 125,132, 137, 136); //on copie un morceau de l'image
imagecopy($destination,$source, 150, 150, 40, 80, 137, 136); //et un autre

imagepng($destination);
imagedestroy($destination, $source);
?>
```

Vous devriez obtenir ceci



imagecopy recopie brutalement l'image. Cette fonction ne s'occupe pas de la transparence. Ce n'est pas grâce à cette fonction que l'on peut fondre deux images. Pour fondre deux images, on utilisera imagecopymerge.

```
imagecopymerge($image_dest, $image_src, $dest_x, $dest_y, $src_x, $src_y, $src_largeur,
$src_hauteur, $taux);
```

Le seul argument inconnu est \$taux. C'est la valeur entière de fusion comprise entre 0 et 100. Si vous n'êtes pas dans l'intervalle indiqué, attendez vous à avoir des résultats bizarres.

#### 22\_imagecopymerge.php

```
<?
header("Content-type: image/png"); //on envoie les infos au navigateur

$source = imagecreatefromjpeg("logo.jpg"); //on ouvre l'image source
$destination = imagecreatefromjpeg("dark_image.jpg");

$details_src = getimagesize("logo.jpg"); //on récupère les dimensions de l'image source

/* on utilise ceci pour calculer l'endroit où on va commencer */
/* à copier. on choisit en bas de l'image : calcul de la */
/* différence de la hauteur de l'image de destination et de */
/* l'image source. */
$y = imagesy($destination)-imagesy($source);

imagecopymerge($destination,$source, 0, $y, 0, 0, $details_src[0],$details_src[1],50); //on copie
l'image

imagepng($destination);
imagedestroy($destination);
imagedestroy($source);
?>
```

Vous devriez obtenir ceci



Le fond blanc est dû à l'image source : le jpeg ne supporte pas la transparence. Vous pouvez fondre une image source en niveaux de gris. Pour cela, utilisez simplement `imagecopymergegray` sur le même schéma que `imagecopymerge`.

### III-3.2.3 - Redimensionner une image

Afin de créer un album photo pratique, vous aurez sans doute besoin de créer des vignettes. Il existe deux fonctions pour réduire les images. La première est `imagecopyresized`.

```
imagecopyresized($image_dest, $image_src, $dest_x, $dest_y, $src_x, $src_y, $dest_largeur, $dest_hauteur, $src_largeur, $src_hauteur);
```

Pour une explication des 6 premiers arguments, reportez vous aux sections précédentes. `$dest_largeur` et `$dest_hauteur` sont les dimensions en pixels de la zone recopiée sur la destination. `$src_largeur` et `$src_hauteur` sont les dimensions en pixels de la zone à copier à partir de la source.

#### 23\_imagecopyresized.php

```
<?php
header('Content-type: image/jpeg');

$ratio = .5;
// Calcul des nouvelles dimensions
list($largeur, $hauteur) = getimagesize("first_open.jpg"); //list est un moyen plus pratique pour ne
récupérer que ce qu'on veut
$n_largeur = $largeur * $ratio;
$n_hauteur = $hauteur * $ratio;

//création de la destination
```

### 23\_imagecopyresized.php

```
$destination = imagecreatetruecolor($n_largeur, $n_hauteur);  
  
//on ouvre la source  
$source = imagecreatefromjpeg("first_open.jpg");  
  
// Redimensionnement  
imagecopyresized($destination, $source, 0, 0, 0, 0, $n_largeur, $n_hauteur, $largeur, $hauteur);  
  
imagejpeg($destination);  
imagedestroy($destination);  
imagedestroy($source);  
?>
```

L'image est réduite mais assombrie et le texte est moins joli. Le plus simple maintenant serait d'utiliser la correction gamma mais comment déterminer le facteur alors qu'on doit en appliquer un différent pour chaque image ? Pas besoin de chercher plus loin, il faut rééchantillonner l'image. On doit donc utiliser l'autre fonction : `imagecopyresampled`. Elle s'emploie exactement comme `imagecopyresized`.

Avec `imagecopyresized`



**Voici votre  
première  
image avec  
GD**

Avec `imagecopyresampled`

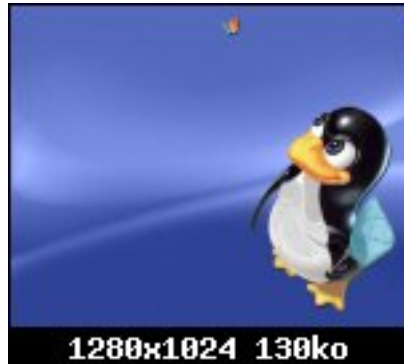


**Voici votre  
première  
image avec  
GD**

## IV - Comment... ? (howto)

### IV-1 - Mettre une légende sous une image

Avoir une image, c'est bien mais parfois, on aime bien avoir des légendes sous ces images. Pour cela, nous avons besoin de connaître les dimensions et le poids de l'image et la largeur des caractères utilisés. Nous allons prendre pour exemple les vignettes produites par [www.imageshack.us](http://www.imageshack.us). Un exemple ci-dessous :



Etapes de programmation

- 1 Récupération du poids de l'image
- 2 Récupération des dimensions
- 3 Calcul des dimensions de l'image à créer
- 4 Création du contour noir
- 5 Création de la légende
- 6 Ouverture du fichier
- 7 Réduction et copie de l'image originale
- 8 Destruction de l'image en mémoire

Vous avez tout ce qu'il faut pour mener à bien ce mini-projet.

#### 25\_vignette.php

```
<?
/*****
/* première étape : récupération du poids de l'image*/
/*****
$file_name = 'tux.jpg';
$file_size = filesize($file_name); //récupération de la taille en octets
$file_size = round($file_size/1024); //conversion en ko

/*****
/* seconde étape : calcul des dimensions de la */
/* nouvelle image comprenant contour + légende */
/*****
list($largeur, $hauteur) = getimagesize($file_name); //dimensions de l'image originale

/*****
/* on veut une image réduite de 150px de large */
/* sans compter les contours */
/* on doit donc diviser la largeur par 150 */
/* on passe à l'inverse car on multiplie par la */
/* suite : cela évite un agrandissement */
/*****

$ratio = 150/ $largeur; //ratio pour réduire à une taille voulue
$vignette_largeur = $largeur * $ratio + 2; //on ajoute 2px à cause du contour
```

## 25\_vignette.php

```

/* on ajoute 3px à cause du contour+légende */
/* imagefontheight retourne la hauteur en pixels */
/* d'une police sélectionnée : cela permet de */
/* calculer la hauteur de la légende */

$vignette_hauteur = $hauteur * $ratio + 3 + imagefontheight(3);

$n_image_largeur = $largeur * $ratio; //largeur de l'image réduite
$n_image_hauteur = $hauteur * $ratio; //hauteur de l'image réduite
/*****
/* création de la vignette : on attribue aucune */
/* couleur pour laisser un cadre noir */
*****/

$image = imagecreatetruecolor($vignette_largeur,$vignette_hauteur);

/*****
/* création de la légende : texte en blanc */
*****/
$blanc = imagecolorallocate($image,255,255,255); //couleur blanche
$string = $largeur."x".$hauteur." ".$file_size."ko"; //création du texte de légende

/* on doit déterminer l'endroit pour commencer */
/* à écrire. On centre le texte d'où la */
/* formulation plus complexe */

$write_h = $vignette_hauteur - imagefontheight(3) - 1; //hauteur

/* centrage -> on récupère $n_largeur la largeur */
/* de la vignette, on lui retire la largeur de la */
/* police multipliée par le nombre de caractères */
/* puis on divise le résultat par 2 pour centrer */

$write_w = ($vignette_largeur - strlen($string) * imagefontwidth(3))/2;
imagestring($image, 3, $write_w, $write_h, $string, $blanc); //on écrit la légende

/*****
/* ouverture du fichier */
*****/
$source = imagecreatefromjpeg($file_name);

/*****
/* réduction, rééchantillonnage et copie de l'image */
/* originale */
/* on recopie l'image à partir du point de */
/* coordonnées 1,1 pour laisser un cadre noir */
*****/

imagecopyresampled($image,$source, 1,1, 0,0, $n_image_largeur , $n_image_hauteur , $largeur,
$hauteur);

/*****
/* envoi de l'image et destruction */
*****/
header('Content-type: image/png');
imagepng($image);
imagedestroy($image);

/*****
/* c'est la fin ! un petit café ? */
*****/
?>

```

Vous devriez obtenir ceci



## IV-2 - Mettre une couleur sur toute l'image

C'est simple, il suffit d'utiliser `imagefill` ou `imagefilledrectangle`.

### ATTENTION ! `imagefill` ne gère plus la transparence depuis gd 2.0.15

26\_over\_color.php

```
<?
//ouverture de l'image
$image = imagecreatefromjpeg('first_open.jpg');

//création d'un rouge transparent
$couleur = imagecolorallocatealpha($image, 255,0,0,50);
//création d'un rectangle rempli de $couleur de la taille de l'image
imagefilledrectangle($image,0,0,imagesx($image),imagesy($image),$couleur);

//envoi des informations au navigateur
header('content-type: image/jpeg');
imagepng($image);
imagedestroy($image);
?>
```

Vous devriez obtenir ceci



## V - Conclusion

Ce tutoriel est fini. J'espère qu'il a été assez clair. Vous avez les notions principales en main pour créer vos propres images. Même si GD ne semble pas utile, il s'avère nécessaire dans de nombreux cas, surtout si vous développez un jeu en ligne ! Je compléterai ce tutoriel dès que possible étant donné que je reprends les cours.

Bon courage et bon dessin