

La manipulation de fichiers en PHP

Introduction

Dans cette partie, nous allons apprendre à manipuler des fichiers en PHP. Nous allons ainsi apprendre à ouvrir et lire un fichier déjà existant ou à créer des fichiers de différents formats (fichiers texte, etc.) grâce aux fonctions PHP et à écrire des informations dedans.

Le PHP met à notre disposition de nombreuses fonctions nous permettant de travailler avec les fichiers car la manipulation de fichiers est un sujet complexe et pour lequel les problématiques peuvent être très différentes et très précises.

Nous n'allons pas dans ce cours explorer le sujet à 100% mais allons plutôt présenter les cas de manipulation de fichiers les plus généraux et les fonctions communes liées.

La manipulation de fichiers en PHP

En PHP, il va être possible de manipuler différents types de fichiers comme des fichiers texte (*au format .txt*) ou des fichiers image par exemple. Bien évidemment, nous n'allons pas pouvoir effectuer les mêmes opérations sur chaque type de fichier.

Ici, nous allons particulièrement nous intéresser à la manipulation de fichiers texte puisque ce sont les fichiers qu'on manipule le plus souvent en pratique et car nous allons pouvoir stocker du texte dans ce type de fichier.

En effet, jusqu'à présent, nous n'avons appris à stocker des informations que de manière temporaire en utilisant les variables « classiques » (*plus tard, nous verrons comment stocker dans les cookies ou les sessions*). L'utilisation de fichiers en PHP va nous permettre, entre autres, de stocker de façon définitive des informations.

Les fichiers vont donc nous offrir une alternative aux bases de données qui servent également à stocker des informations de manière organisée et définitive et que nous étudierons plus tard même s'il faut bien admettre qu'on préférera dans la majorité des cas utiliser les bases de données plutôt que les fichiers pour enregistrer les données.

Il reste néanmoins intéressant d'apprendre à stocker des données dans des fichiers car il sera préférable dans certains cas de stocker des données dans des fichiers plutôt que dans des bases de données (*par exemple lorsqu'on laissera la possibilité aux utilisateurs de télécharger ensuite le fichier avec ses différentes informations à l'intérieur*).

Par ailleurs, la manipulation de fichiers ne se limite pas au stockage de données : il sera possible d'effectuer toutes sortes d'opérations sur les fichiers ce qui représente un avantage indéniable.

Lire un fichier entier en PHP

L'une des opérations de base relative aux fichiers en PHP va tout simplement être de lire le contenu d'un fichier.

Il existe deux façons de faire cela :

1. lire le contenu d'un fichier morceau par morceau ;
2. lire un fichier entièrement c'est-à-dire afficher tout son contenu d'un coup.

Pour faire cela, on va pouvoir utiliser l'une des fonctions suivantes :

- La fonction `file_get_contents()` ;
- La fonction `file()` ;
- La fonction `readfile()` ;

On va devoir passer le chemin relatif menant au fichier qu'on souhaite lire en argument de chacune de ces fonctions.

La fonction `file_get_contents()` va retourner le contenu du fichier dans une chaîne de caractères qu'il faudra `echo` pour afficher ou la valeur booléenne false en cas d'erreur.

La fonction `file()` est identique à `file_get_contents()` à la seule différence que le contenu du fichier sera renvoyé dans un tableau numéroté. Chaque ligne de notre fichier sera un nouvel élément de tableau.

La fonction `readfile()` va elle directement lire et afficher le contenu de notre fichier.

Pour illustrer le fonctionnement de ces trois fonctions, je vous invite à créer un premier fichier de texte, c'est-à-dire un fichier enregistré avec l'extension .txt.

Pour le test, utilisons `exemple.txt` et qui contient le texte suivant :

```
1 Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers
2 en PHP.
3
4 Nous allons ainsi apprendre à ouvrir et lire un fichier déjà
5 existant ou à créer des fichiers de différents formats (fichiers texte, etc.)
6 grâce aux fonctions PHP et à écrire des informations dedans.
```

Pour créer ce fichier texte, servez-vous de votre éditeur et enregistrez-le dans le même dossier que votre fichier `cours.php` (pour plus de facilité).

Dès que le fichier est créé et enregistré, nous allons tenter de l'afficher dans notre script :

```
<!DOCTYPE html>
```

```
<html>
<head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="cours.css">
</head>

<body>
    <h1>Titre principal</h1>
    <?php
        echo file_get_contents('test.txt');
        echo '<br><br>';

        echo '<pre>';
        print_r(file('test.txt'));
        echo '</pre>';
        echo '<br><br>';

        readfile('test.txt');
    ?>
    <p>Un paragraphe</p>
</body>
</html>
```

Titre principal

Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers en PHP. Nous allons ainsi apprendre à ouvrir et lire un fichier déjà existant ou à créer des fichiers de différents formats (fichiers texte, etc.) grâce aux fonctions PHP et à écrire des informations dedans.

```
Array
(
    [0] => Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers
    [1] => en PHP.
    [2] =>
    [3] => Nous allons ainsi apprendre à ouvrir et lire un fichier déjà
    [4] => existant ou à créer des fichiers de différents formats (fichiers texte, etc.)
    [5] => grâce aux fonctions PHP et à écrire des informations dedans.
)
```

Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers en PHP. Nous allons ainsi apprendre à ouvrir et lire un fichier déjà existant ou à créer des fichiers de différents formats (fichiers texte, etc.) grâce aux fonctions PHP et à écrire des informations dedans.

Un paragraphe

Dans l'exemple ci-dessus, nous utilisons les trois fonctions `file_get_contents()`, `file()` et `readfile()` à la suite pour lire et afficher le contenu du fichier .txt. Comme vous pouvez le voir, `file_get_contents()` et `readfile()` produisent un résultat similaire mais il faut echo le résultat renvoyé par `file_get_contents()` pour l'afficher tandis que `readfile()` affiche directement le résultat.

La fonction `file_get_contents()` nous donne donc davantage de contrôle sur la valeur renvoyée puisqu'on va pouvoir la stocker dans une variable pour la manipuler à loisir.

Conserver la mise en forme d'un fichier avant affichage

Dans l'exemple ci-dessus, on s'aperçoit que les retours à la ligne et les sauts de ligne contenus dans notre fichier texte ne sont pas conservés lors de l'affichage du texte dans le navigateur.

Cela est normal puisque nos fonctions vont renvoyer le texte tel quel sans y ajouter des éléments HTML indiquant de nouvelles lignes ou des sauts de ligne et donc le navigateur va afficher le texte d'une seule traite.

Pour conserver la mise en forme du texte, on va pouvoir utiliser la fonction `nl2br()` qui va se charger d'ajouter des éléments `
` devant chaque nouvelle ligne de notre texte. On va passer le texte à mettre en forme en argument de cette fonction.

On va par exemple pouvoir utiliser cette fonction sur le résultat renvoyé par `file_get_contents()` avant de l'afficher. En revanche, on ne va pas pouvoir l'utiliser avec `readfile()` puisque cette fonction va directement afficher le résultat sans que nous puissions exercer un quelconque contrôle dessus.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
        echo nl2br(file_get_contents('test.txt'));
        echo '<br><br>';

        echo '<pre>';
        print_r(file('test.txt'));
        echo '</pre>';
        echo '<br><br>';

        readfile('test.txt');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Titre principal

Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers en PHP.

Nous allons ainsi apprendre à ouvrir et lire un fichier déjà existant ou à créer des fichiers de différents formats (fichiers texte, etc.) grâce aux fonctions PHP et à écrire des informations dedans.

```
array
{
    [0] => Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers
    [1] => en PHP.
    [2] =>
    [3] => Nous allons ainsi apprendre à ouvrir et lire un fichier déjà
    [4] => existant ou à créer des fichiers de différents formats (fichiers texte, etc.)
    [5] => grâce aux fonctions PHP et à écrire des informations dedans.
}
```

Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers en PHP. Nous allons ainsi apprendre à ouvrir et lire un fichier déjà existant ou à créer des fichiers de différents formats (fichiers texte, etc.) grâce aux fonctions PHP et à écrire des informations dedans.

Un paragraphe

Ici, on effectue plusieurs opérations en une seule ligne. Comme d'habitude, lorsqu'on utilise plusieurs fonctions ensemble, c'est la fonction la plus interne du code qui va s'exécuter en premier puis la fonction qui l'englobe va s'exécuter ensuite sur le résultat de la première fonction.

Dans le cas présent, `file_get_contents()` s'exécute en premier et renvoie le contenu du fichier puis `nl2br()` s'exécute ensuite sur le résultat renvoyé par `file_get_contents()` (*c'est-à-dire le texte du fichier .txt*) et ajoute des éléments `
` à chaque nouveau retour à la ligne. Finalement, la structure de langage `echo` permet d'afficher le résultat (*le texte avec nos éléments
*).

Ouvrir, lire et fermer un fichier en PHP

Ci-dessus, nous avons appris à lire un fichier texte entièrement. Souvent, nous ne voudrons lire et récupérer qu'une partie d'un fichier. Alors, comment faire ?.

Ouvrir un fichier en PHP

Pour lire un fichier partie par partie, nous allons avant tout devoir l'ouvrir. Pour ouvrir un fichier en PHP, nous allons utiliser la fonction **fopen()**, abréviation de « *file open* » ou « *ouverture de fichier* » en français.

On va devoir passer le chemin relatif menant à au fichier ainsi qu'un « *mode* » en arguments de cette fonction qui va alors retourner (*en cas de succès*) une ressource de pointeur de fichier, c'est-à-dire pour le dire simplement : une ressource qui va permettre de manipuler le fichier.

Le mode choisi détermine le type d'accès au fichier et donc les opérations qu'on va pouvoir effectuer sur le fichier. On va pouvoir choisir parmi les modes suivants :

Mode d'ouverture	Description
r	Ouvre un fichier en lecture seule. Il est impossible de modifier le fichier.
r+	Ouvre un fichier en lecture et en écriture.
a	Ouvre un fichier en écriture seule en conservant les données existantes. Si le fichier n'existe pas, le PHP tente de le créer.
a+	Ouvre un fichier en lecture et en écriture en conservant les données existantes. Si le fichier n'existe pas, le PHP tente de le créer.
w	Ouvre un fichier en écriture seule. Si le fichier existe, les informations existantes seront supprimées. S'il n'existe pas, crée un fichier.
w+	Ouvre un fichier en lecture et en écriture. Si le fichier existe, les informations existantes seront supprimées. S'il n'existe pas, crée un fichier.
x	Crée un nouveau fichier accessible en écriture seulement. Retourne false et une erreur si le fichier existe déjà.

x+	Crée un nouveau fichier accessible en lecture et en écriture. Retourne false et une erreur si le fichier existe déjà.
c	Ouvre un fichier pour écriture seulement. Si le fichier n'existe pas, il sera créé. Si il existe, les informations seront conservées.
c+	Ouvre un fichier pour lecture et écriture. Si le fichier n'existe pas, il sera créé. Si il existe, les informations seront conservées.
e	Le mode e est particulier et n'est pas toujours disponible. Nous n'en parlerons pas ici.

Comme vous pouvez le constater, le choix du mode influe fortement sur les opérations que nous allons pouvoir réaliser sur le fichier en question.

Par exemple, lorsqu'on ouvre un fichier en lecture seulement toute modification de ce fichier est impossible, ce qui n'est pas le cas si on choisit un mode permettant l'écriture dans ce fichier.

Notez par ailleurs qu'on ajoutera généralement la lettre **b** au paramètre « mode » de **fopen()**. Cela permet une meilleure compatibilité et évite les erreurs pour les systèmes qui différencient les fichiers textes et binaires comme Windows par exemple.

Lire un fichier partie par partie

PHP met à notre disposition plusieurs fonctions qui vont nous permettre de lire un fichier partie par partie :

- La fonction **fread()** ;
- La fonction **fgets()** ;
- La fonction **fgetc()**.

Lire un fichier jusqu'à un certain point avec fread()

La fonction **fread()** (*abréviation de « file read » ou « lecture de fichier » en français*) va prendre en arguments le fichier renvoyé par **fopen()** ainsi qu'un nombre qui correspond aux nombres d'octets du fichier qui doivent être lus.

Pour lire le fichier entièrement, on peut utiliser la fonction **filesize()** en lui passant le nom du fichier qu'on souhaite lire en deuxième argument de **fread()**. En effet, **filesize()** renvoie la

Exemple :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="cours.css">
  </head>

  <body>
    <h1>Titre principal</h1>
    <?php
      $ressource = fopen('test.txt', 'rb');
      echo fread($ressource, filesize('test.txt'));
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

Titre principal

Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers en PHP. Nous allons ainsi apprendre à ouvrir et lire un fichier déjà existant ou à créer des fichiers de différents formats (fichiers texte, etc.) grâce aux fonctions PHP et à écrire des informations dedans.

Un paragraphe

Ici, on commence par utiliser **fopen()** pour lire notre fichier et on récupère la ressource renvoyée par la fonction dans une variable **\$ressource**.

On passe ensuite le contenu de notre variable à **fread()** et on lui demande de lire le fichier jusqu'au bout en lui passant la taille exacte du fichier obtenue avec **filesize()**.

Finalement, on affiche le résultat renvoyé par **fread()** grâce à une structure **echo**.

Lire un fichier ligne par ligne avec fgets()

La fonction **fgets()** va nous permettre de lire un fichier ligne par ligne. On va passer le résultat renvoyé par **fopen()** en argument de **fgets()** et à chaque nouvel appel de la fonction, une nouvelle ligne du fichier va pouvoir être lue.

On peut également préciser de manière facultative un nombre en deuxième argument de **fgets()** qui représente un nombre d'octets. La fonction lira alors soit le nombre d'octets précisé, soit jusqu'à la fin du fichier, soit jusqu'à arriver à une nouvelle ligne (*le premier des*

trois cas qui va se présenter). Si aucun nombre n'est précisé, **fgets()** lira jusqu'à la fin de la ligne.

Notez que si on précise un nombre d'octets maximum à lire inférieur à la taille de notre ligne et qu'on appelle successivement **fgets()**, alors la fin de la ligne courante sera lue lors du deuxième appel de **fgets()**.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $ressource = fopen('test.txt', 'rb');

            //Lit les 30 premiers caractères du fichier
            echo 'Premier appel : ' . fgets($ressource, 30). '<br>';

            //Lit le reste de la première ligne
            echo 'Deuxième appel : ' . fgets($ressource). '<br>';

            //Lit la deuxième ligne du fichier
            echo 'Troisième appel : ' . fgets($ressource). '<br>';
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Titre principal

Premier appel : Dans cette nouvelle section,
Deuxième appel : nous allons apprendre à manipuler des fichiers
Troisième appel : en PHP.

Un paragraphe

Lire un fichier caractère par caractère avec getc()

Dans certains cas, il va également être intéressant de lire un fichier caractère par caractère notamment pour récupérer un caractère en particulier ou pour arrêter la lecture lorsqu'on arrive à un certain caractère.

Pour faire cela, nous allons cette fois-ci utiliser la fonction **fgetc()**. Cette fonction va s'utiliser exactement comme **fgets()**, et chaque nouvel appel à la fonction va permettre de lire un

nouveau caractère du fichier. Notez que les espaces sont bien entendus considérés comme des caractères.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="cours.css">
  </head>

  <body>
    <h1>Titre principal</h1>
    <?php
      $ressource = fopen('test.txt', 'rb');

      //Lit le premier caractère du fichier
      echo 'Premier appel : ' . fgetc($ressource) . '<br>';

      //Lit le deuxième caractère
      echo 'Deuxième appel : ' . fgetc($ressource) . '<br>';

      //Lit le troisième caractère
      echo 'Troisième appel : ' . fgetc($ressource) . '<br>';
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

Titre principal

Premier appel : D
Deuxième appel : a
Troisième appel : n

Un paragraphe

Trouver la fin d'un fichier de taille inconnue

Jusqu'ici, nous avons travaillé sur des exemples simples. Ici, par exemple, nous avons nous mêmes créé un fichier pour effectuer différentes opérations dessus.

En pratique, cependant, nous utiliserons généralement les fichiers pour stocker des informations non connues à l'avance. Dans ce cas-là, il est impossible de prévoir à l'avance la taille de ces fichiers et on risque donc de ne pas pouvoir utiliser les fonctions **fgets()** et **fgetc()** de manière optimale.

Il existe plusieurs moyens de déterminer la taille ou la fin d'un fichier. La fonction **filesize()**, par exemple, va lire la taille d'un fichier. Dans le cas présent, cependant, nous cherchons plutôt à déterminer où se situe la fin d'un fichier (*ce qui n'est pas forcément équivalent à la taille d'un fichier à cause de la place du curseur, notion que nous allons voir en détail par la suite*).

La fonction PHP `feof()` (« *eof* » signifie « *end of the file* » ou « *fin du fichier* ») va nous permettre de savoir si la fin d'un fichier a été atteinte ou pas. Dès que la fin d'un fichier est atteinte, cette fonction va renvoyer la valeur `true`. Avant cela, elle renverra la valeur `false`. On va donc pouvoir utiliser cette fonction pour boucler dans un fichier de taille inconnue.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $res = fopen('test.txt', 'rb');

            /*Tant que la fin du fichier n'est pas atteinte, c'est-à-dire
             *tant que feof() renvoie FALSE (= tant que !feof() renvoie TRUE)
             *on echo une nouvelle ligne du fichier*/
            while(!feof($res)){
                $ligne = fgets($res);
                echo 'La ligne "' . $ligne. '" contient
                    ' . strlen($ligne). ' caractères <br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Titre principal

La ligne "Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers " contient 77 caractères

La ligne "en PHP." contient 8 caractères

La ligne " " contient 1 caractères

La ligne "Nous allons ainsi apprendre à ouvrir et lire un fichier déjà " contient 64 caractères

La ligne "existant ou à créer des fichiers de différents formats (fichiers texte, etc.) " contient 81 caractères

La ligne "grâce aux fonctions PHP et à écrire des informations dedans." contient 63 caractères

Un paragraphe

Ici, tant que la fin du fichier n'est pas atteinte, on affiche une nouvelle ligne du fichier et on calcule le nombre de caractères de la ligne grâce à la fonction `strlen()`, abréviation de « *string length* » ou « *longueur de la chaîne* » en français.

Notez qu'ici le fait de retourner à la ligne compte comme un caractère et que la fonction `fgets()` s'arrête après ce passage à la ligne. C'est la raison pour laquelle, lorsqu'on compte le nombre de caractères de nos lignes, on a un caractère de plus que ce à quoi on pouvait s'attendre (*sauf pour la dernière*).

La place du curseur interne ou pointeur de fichier

La position du curseur (*ou « pointeur de fichier »*) va impacter le résultat de la plupart des manipulations qu'on va pouvoir effectuer sur les fichiers. Il est donc essentiel de toujours savoir où se situe ce pointeur et également de savoir comment le bouger.

Le curseur ou pointeur est l'endroit dans un fichier à partir duquel une opération va être faite. Pour donner un exemple concret, le curseur dans un document Word, dans un champ de formulaire ou lorsque vous effectuez une recherche Google ou tapez une URL dans votre navigateur correspond à la barre clignotante.

Ce curseur indique l'emplacement à partir duquel vous allez écrire votre requête ou supprimer un caractère, etc. Le curseur dans les fichiers va être exactement la même chose à la différence qu'ici on ne peut pas le voir visuellement.

Le mode d'ouverture choisi va être la première chose qui va influer sur la position du pointeur. En effet, selon le mode choisi, le pointeur de fichier va se situer à une place différente et va pouvoir ou ne va pas pouvoir être déplacé :

Mode utilisé	Position du pointeur de fichier
r / r+	Début du fichier
a / a+	Fin du fichier
w / w+	Début du fichier
x / x+	Début du fichier
c / c+	Début du fichier

Ensuite, vous devez également savoir que certaines fonctions vont modifier la place du curseur à chaque exécution. Cela va par exemple être le cas des fonctions **fgets()** et **fgetc()** qui servent à lire un fichier ligne par ligne ou caractère par caractère.

En effet, la première fois qu'on appelle **fgets()** par exemple, le pointeur est généralement au début de notre fichier et c'est donc la première ligne de notre fichier est lue par défaut. Cependant, lors du deuxième appel à cette fonction, c'est bien la deuxième ligne de notre fichier qui va être lue.

Ce comportement est justement dû au fait que la fonction **fgets()** déplace le pointeur de fichier du début de la première ligne au début de la seconde ligne dans ce cas précis.

Pour savoir où se situe notre pointeur de fichier, on peut utiliser la fonction **f.tell()** qui renvoie la position courante du pointeur. Nous allons devoir lui passer la valeur renvoyée par **f.open()** pour qu'elle fonctionne correctement.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $res = fopen('test.txt', 'rb');

            while(!feof($res)) {
                echo 'Le pointeur est au niveau du caractère ' . ftell($res) . '<br>';
                $ligne = fgets($res);
                echo 'La ligne "' . $ligne. '" contient
                    ' . strlen($ligne). ' caractères <br><br>';
            }
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Titre principal

Le pointeur est au niveau du caractère 0

La ligne "Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers " contient 77 caractères

Le pointeur est au niveau du caractère 77
La ligne "en PHP." contient 8 caractères

Le pointeur est au niveau du caractère 85
La ligne " " contient 1 caractères

Le pointeur est au niveau du caractère 86
La ligne "Nous allons ainsi apprendre à ouvrir et lire un fichier déjà " contient 64 caractères

Le pointeur est au niveau du caractère 150
La ligne "existant ou à créer des fichiers de différents formats (fichiers texte, etc.) " contient 81 caractères

Le pointeur est au niveau du caractère 231
La ligne "grâce aux fonctions PHP et à écrire des informations dedans." contient 63 caractères

Un paragraphe

Déplacer le curseur manuellement

Pour commencer la lecture d'un fichier à partir d'un certain point, ou pour écrire dans un fichier à partir d'un endroit précis ou pour toute autre manipulation de ce type, nous allons avoir besoin de contrôler la position du curseur. Pour cela, nous allons pouvoir utiliser la fonction **fseek()**.

Cette fonction va prendre en arguments l'information renvoyée par **fopen()** ainsi qu'un nombre correspondant à la nouvelle position en octets du pointeur.

La nouvelle position du pointeur sera par défaut calculée par rapport au début du fichier. Pour modifier ce comportement et faire en sorte que le nombre passé s'ajoute à la position courante du curseur, on peut ajouter la constante **SEEK_CUR** en troisième argument de **fseek()**.

Notez cependant que si vous utilisez les modes **a** et **a+** pour ouvrir votre fichier, utiliser la fonction **fseek()** ne produira aucun effet et votre curseur se placera toujours en fin de fichier.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
  </head>

  <body>
    <h1>Titre principal</h1>
    <?php
      $res = fopen('test.txt', 'rb');

      echo 'Le pointeur est derrière le caractère ' .ftell($res). '<br>';
      echo 'Le caractère ' .(ftell($res) + 1). ' est un ' .fgetc($res). '<br>';
      echo 'Le pointeur est derrière le caractère ' .ftell($res). '<br>';
      fseek($res, 20);
      echo 'Le pointeur est derrière le caractère ' .ftell($res). '<br>';
      echo 'Le caractère ' .(ftell($res) + 1). ' est un ' .fgetc($res). '<br>';
      echo 'Le pointeur est derrière le caractère ' .ftell($res). '<br>';
      fseek($res, 40, SEEK CUR);
      echo 'Le pointeur est derrière le caractère ' .ftell($res). '<br>';
      echo 'Le caractère ' .(ftell($res) + 1). ' est un ' .fgetc($res). '<br>';
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

Titre principal

Le pointeur est derrière le caractère 0
Le caractère 1 est un D
Le pointeur est derrière le caractère 1
Le pointeur est derrière le caractère 20
Le caractère 21 est un s
Le pointeur est derrière le caractère 21
Le pointeur est derrière le caractère 61
Le caractère 62 est un e

Un paragraphe

Fermer un fichier

Pour fermer un fichier en PHP, nous utiliserons cette fois la fonction **fclose()**.

On va une nouvelle fois passer le résultat renvoyé par **fopen()** en argument de cette fonction.

Notez que la fermeture d'un fichier n'est pas strictement obligatoire. Cependant, cela est considéré comme une bonne pratique : cela évite d'user inutilement les ressources de votre serveur.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $res = fopen('test.txt', 'rb');

            echo fread($res, filesize('test.txt'));
            fclose($res);
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

Titre principal

Dans cette nouvelle section, nous allons apprendre à manipuler des fichiers en PHP. Nous allons ainsi apprendre à ouvrir et lire un fichier déjà existant ou à créer des fichiers de différents formats (fichiers texte, etc.) grâce aux fonctions PHP et à écrire des informations dedans.

Un paragraphe

Créer et écrire dans un fichier en PHP

Nous avons appris à ouvrir un fichier et avons découvert l'importance du mode d'ouverture qui va conditionner les opérations qu'on va pouvoir effectuer sur le fichier ouvert.

Nous avons également appris à gérer la place du pointeur, ce qui va se révéler essentiel pour écrire dans un fichier en PHP contenant déjà du texte.

A présent, nous allons apprendre à créer un fichier et à écrire dans un fichier vierge ou contenant déjà du texte.

Créer et écrire dans un fichier en PHP : les méthodes disponibles

Il existe différentes façons de créer un fichier et d'écrire dans un fichier déjà existant ou pas et contenant déjà du texte ou pas en PHP.

Les deux façons les plus simples vont être d'utiliser :

- soit la fonction `file_put_contents()` ;
- soit les fonctions `fopen()` et `fwrite()` ensemble.

Notez déjà que l'utilisation des fonctions `fopen()` et `fwrite()` va nous donner plus de contrôle sur notre écriture, en nous permettant de choisir un mode d'ouverture, d'écrire à un endroit du fichier, etc...

Écrire dans un fichier avec `file_put_contents()`

La fonction `file_put_contents()` va nous permettre d'écrire simplement des données dans un fichier.

Cette fonction va accepter en argument le chemin vers le fichier dans lequel on doit écrire les données, les données à écrire (*qui peuvent être une chaîne de caractères ou un tableau*) ainsi qu'un flag "drapeau" (*nous reviendrons là-dessus plus tard*).

Si le fichier spécifié dans le chemin du fichier n'existe pas, alors il sera créé. S'il existe, il sera par défaut écrasé, ce qui signifie que ses données seront supprimées.

Vous pouvez déjà noter qu'appeler `file_put_contents()` correspond à appeler successivement les fonctions `fopen()`, `fwrite()` et `fclose()`.

Utilisons immédiatement cette fonction pour écrire dans un premier fichier :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="cours.css">
  </head>

  <body>
    <h1>Titre principal</h1>
    <?php
      file_put_contents('exemple.txt', 'Ecriture dans un fichier');
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

1 Ecriture dans un fichier

Ici, on passe le chemin de fichier exemple.txt à la fonction `file_put_contents()`. On va donc chercher un fichier qui s'appelle exemple.txt et qui se situe dans le même répertoire que notre script.

Comme ce fichier n'existe pas, il va être créé et le texte précisé en deuxième argument de notre fonction va être inséré dedans.

Si on essaie maintenant de répéter l'opération et d'écrire un texte différent dans notre fichier, on s'aperçoit que le nouveau texte remplace l'ancien qui est écrasé. C'est le comportement par défaut :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Titre principal</h1>
    <?php
      file_put_contents('exemple.txt', 'Ecriture dans un fichier');
      file_put_contents('exemple.txt', '**NOUVEAU TEXTE**');
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

1 **NOUVEAU TEXTE**

Pour « ajouter » du texte à notre fichier, une astuce simple consiste ici à récupérer le texte d'origine de notre fichier dans une variable, puis à le concaténer avec le texte qu'on souhaite écrire dans notre fichier, puis à passer le nouveau texte à `file_put_contents()`. L'ancien contenu du fichier sera alors remplacé par le nouveau comme précédemment.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Titre principal</h1>
    <?php
      //On écrit un premier texte dans notre fichier
      file_put_contents('exemple.txt', 'Ecriture dans un fichier');

      //On récupère le contenu du fichier
      $texte = file_get_contents('exemple.txt');

      //On ajoute notre nouveau texte à l'ancien
      $texte .= "\n**NOUVEAU TEXTE**";

      //On écrit tout le texte dans notre fichier
      file_put_contents('exemple.txt', $texte);
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

```
1 Ecriture dans un fichier
2 **NOUVEAU TEXTE**
```

Ici, vous pouvez noter qu'est utilisé le caractère `\n`. Celui-ci sert à créer un retour à la ligne en PHP. Ici, on l'utilise pour que notre nouveau texte soit inséré dans la ligne suivante de notre fichier. On est obligés d'utiliser des guillemets plutôt que des apostrophes ici pour que le `\n` soit bien interprété comme un retour à la ligne par le PHP.

Cette astuce pour ajouter du texte dans un fichier fonctionne mais nous force finalement à faire plusieurs opérations et à écraser le contenu de base du fichier pour placer le nouveau (*qui contient le contenu original*).

Pour maintenant véritablement conserver les données de base de notre fichier et lui ajouter de nouvelles données à la suite des données déjà précédentes, on va plus simplement pouvoir passer le flag `FILE_APPEND` en troisième argument de notre fonction `file_put_contents()`.

Un flag (*drapeau*) est une constante qui va correspondre à un nombre. De nombreuses fonctions utilisent des flags différents. Le flag ou la constante `FILE_APPEND` va ici nous permettre d'ajouter des données en fin de fichier (*c'est-à-dire à la suite du texte déjà existant*) plutôt que d'écraser les données déjà existantes.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
  </head>

  <body>
    <h1>Titre principal</h1>
    <?php
      //On écrit un premier texte dans notre fichier
      file_put_contents('exemple.txt', 'Ecriture dans un fichier');

      //On rajoute du texte dans notre fichier
      file_put_contents('exemple.txt', "\n**NOUVEAU TEXTE**", FILE_APPEND);
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

1 Ecriture dans un fichier

2 **NOUVEAU TEXTE**

Ici, on utilise une première fois `file_put_contents()` sans le drapeau `FILE_APPEND` pour écrire “*Ecriture dans un fichier dans notre fichier*”. Ce texte va donc écraser les données déjà présentes dans le fichier. Ensuite, on ajoute un autre texte en utilisant le flag.

Créer un fichier PHP avec fopen()

Pour créer un nouveau fichier en PHP (*sans forcément écrire dedans*), nous allons à nouveau utiliser la fonction `fopen()`. En effet, rappelez-vous que cette fonction va pouvoir créer un fichier si celui-ci n'existe pas à condition qu'on utilise un mode adapté.

Pour créer un fichier avec `fopen()`, nous allons devoir lui passer en arguments un nom de fichier qui sera le nom du fichier créé ainsi qu'un mode d'ouverture adapté. En mentionnant simplement cela, le fichier sera par défaut créé dans le même dossier que le script PHP.

Créons un fichier qu'on va appeler `exemple2.txt`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
  </head>

  <body>
    <h1>Titre principal</h1>
    <?php
      $fichier = fopen('exemple2.txt', 'c+b');
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

Ici, nous créons un nouveau fichier avec le mode **c+**. Le fichier est donc accessible en lecture et en écriture et si celui-ci existait et contenait déjà des informations, celles-ci ne seraient pas effacées à la différence du mode **w+**.

On pense bien également à rajouter l'option **b** (pour binaire) afin de maximiser la compatibilité pour les différents systèmes.

Bien évidemment, afin de véritablement créer le fichier, le code de votre page doit être exécuté.

Une fois cela fait, vous pouvez aller vérifier dans le dossier contenant votre page PHP qu'un fichier nommé exemple2.txt a bien été créé.

Écrire dans un fichier avec fwrite()

Une fois un fichier ouvert ou créé avec **fopen()**, on va pouvoir écrire dedans en utilisant la fonction **fwrite()**.

Cette fonction va prendre la valeur retournée par **fopen()** ainsi que la chaîne de caractères à écrire dans le fichier en arguments.

Si le fichier est vide, on va très simplement pouvoir écrire du texte dedans :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours PHP & MySQL</title>
        <meta charset="utf-8">
    </head>

    <body>
        <h1>Titre principal</h1>
        <?php
            $fichier = fopen('exemple2.txt', 'c+b');
            fwrite($fichier, 'Un premier texte dans mon fichier');
        ?>
        <p>Un paragraphe</p>
    </body>
</html>
```

1 Un premier texte dans mon fichier

Dans le cas où le fichier ouvert contient déjà du texte, cela va être un peu plus complexe. En effet, il va déjà falloir se poser la question d'où se situe le pointeur.

Si on utilise la fonction **fwrite()** plusieurs fois de suite, le texte va être ajouté à la suite car **fwrite()** va déplacer le pointeur après le texte inséré après chaque utilisation.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
      <link rel="stylesheet" href="cours.css">
  </head>

  <body>
    <h1>Titre principal</h1>
    <?php
      $fichier = fopen('exemple2.txt', 'c+b');
      fwrite($fichier, 'Un premier texte dans mon fichier');
      fwrite($fichier, '. Un autre texte');
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

```
1 Un premier texte dans mon fichier. Un autre texte
```

Le problème va se situer lors de la première utilisation de `fwrite()` dans un fichier qui contient déjà du texte. En effet, la plupart des modes de `fopen()` vont placer le curseur en début de fichier. Les informations vont donc être écrites par-dessus les anciennes.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
      <link rel="stylesheet" href="cours.css">
  </head>

  <body>
    <h1>Titre principal</h1>
    <?php
      /*Notre fichier contient toujours le texte :
       *"Un premier texte dans mon fichier. Un autre texte"
       *ajouté précédemment*/
      $fichier = fopen('exemple2.txt', 'c+b');
      fwrite($fichier, 'abc');
      fwrite($fichier, 'def');
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

1 abcdefmier texte dans mon fichier. Un autre texte

Ici, le fichier contient le texte « *Un premier texte dans mon fichier. Un autre texte* » lors de son ouverture. Lors du premier appel à **fwrite()**, le curseur se situe au début du fichier. Les informations « *abc* » vont alors être écrites par-dessus celles déjà présentes.

Après sa première utilisation, **fwrite()** déplace le curseur à la fin du texte ajouté, c'est-à-dire juste derrière le caractère « *c* ». Si on appelle **fwrite()** à nouveau immédiatement après, les nouvelles informations vont être insérées après le « *c* ».

On va ici pouvoir utiliser la fonction **fseek()** pour modifier la position du pointeur et écrire à partir d'un autre endroit dans le fichier. En faisant cela, le nouveau texte sera écrit à partir d'un certain point dans le fichier. Si on tente d'écrire du texte au milieu du fichier, cependant, les données déjà présentes à cet endroit continueront d'être écrasées.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
  </head>

  <body>
    <h1>Titre principal</h1>
    <?php
      /*Notre fichier contient toujours le texte :
       *"abcdefmier texte dans mon fichier. Un autre texte"
       *ajouté précédemment*/
      $fichier = fopen('exemple2.txt', 'c+b');
      fwrite($fichier, 'abc');
      fwrite($fichier, 'def');

      //On déplace le curseur de 20 octets
      fseek($fichier, 20, SEEK_CUR);
      fwrite($fichier, 'ghijk');

      //On place le curseur en fin de fichier
      fseek($fichier, filesize('exemple2.txt'));
      fwrite($fichier, 'lmnop');
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

1 abcdefmier texte dans mon ghijk. Un autre textelmnop

Si on souhaite ajouter du texte au milieu d'un fichier tout en conservant le texte précédent, nous allons devoir procéder en plusieurs étapes.

L'idée va être ici de récupérer la première partie du texte de notre fichier jusqu'au point d'insertion du nouveau contenu, puis de concaténer le nouveau contenu à ce texte, puis de

récupérer la deuxième partie du texte de notre fichier et de la concaténer au reste avant de finalement écrire le tout dans notre fichier.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours PHP & MySQL</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Titre principal</h1>
    <?php
      /*On appelle fopen() avec le mode c : le pointeur se situe
       *au début du fichier. Le fichier contient le texte :
       *"abcdefmier texte dans mon ghijker. Un autre textelmnop"*/
      $fichier = fopen('exemple2.txt', 'c+b');

      /*On lit les 20 premiers octets du fichier avec fread(), le
       *pointeur se situe là où fread() arrête sa lecture*/
      $texte = fread($fichier, 20);

      //On ajoute du texte dans notre variable
      $texte .= ' TEXTE AJOUTE AU MILIEU ';

      /*On lit la suite du fichier (fread() reprend sa lecture là où se
       *trouve le pointeur) et on ajoute le texte lu dans $texte*/
      $texte .= fread($fichier, filesize('exemple2.txt'));

      /*On replace le pointeur en début de fichier et on écrase l'ancien
       *texte avec le nouveau (qui est plus long)*/
      fseek($fichier, 0);
      fwrite($fichier, $texte);
    ?>
    <p>Un paragraphe</p>
  </body>
</html>
```

1 abcdefmier texte dan TEXTE AJOUTE AU MILIEU s mon ghijker. Un autre textelmnop