La manipulation de fichiers en PHP

Table des matières

I. Contexte 3

II. Manipuler des dossiers 3

III. Exercice : Appliquez la notion 6

IV. Manipuler des fichiers 7

V. Exercice : Appliquez la notion 9

VI. Lire un fichier 9

VII. Exercice : Appliquez la notion 11

VIII. Écrire dans un fichier 12

IX. Exercice : Appliquez la notion 13

X. Lire les métadonnées 14

XI. Exercice : Appliquez la notion 16

XII. Auto-évaluation 17

A. Exercice final 17

B. Exercice : Défi 18

Solutions des exercices 19

I. Contexte

Durée : 1 h

Environnement de travail : repl.it

Pré-requis : Connaître les bases de PHP et la gestion des droits de fichiers

Contexte

Certains cas d'utilisation d'une application PHP demanderont de manipuler des fichiers présents sur le serveur, que ce soit pour de l'enregistrement, de la consultation ou de la modification de fichiers.

PHP offre de nombreuses fonctionnalités permettant la gestion de fichiers.

Nous verrons ici comment il nous permet de manipuler des dossiers, puis nous étudierons l'ouverture et la fermeture de fichiers, avant de décrire les fonctionnalités permettant tout d'abord de lire, puis d'écrire dans un fichier.

Enfin, nous présenterons quelques fonctions de récupération des informations d'un fichier.

II. Manipuler des dossiers

Objectifs

- Comprendre la notion de chemins relatifs et absolus
- Apprendre à créer un dossier, le parcourir et le supprimer

Mise en situation

Afin de pouvoir réaliser la gestion des fichiers présents sur notre système, il faut pouvoir accéder à leurs positions dans l'arborescence des fichiers, et donc parcourir les différents répertoires pour les atteindre.

Ce chapitre détaillera les notions des chemins relatif et absolu, abordera comment créer un répertoire, avant de montrer que ceux-ci peuvent être parcourus afin d'en récupérer le contenu.

Chemins relatif et absolu

Il existe deux façons de décrire le chemin d'accès d'une ressource sur le serveur : de manière relative ou absolue.

Définition **Chemin relatif**

Un chemin d'accès relatif se décrit en partant du répertoire courant jusqu'à atteindre la ressource cible.

Un **chemin relatif** commence directement par un nom de ressource ou est précédé de `./` pour cibler le répertoire courant, ou de `../` pour remonter d'un niveau de répertoire.

Exemple

```
../../images/icon.jpg
```

Le chemin d'accès relatif ci-dessus cible le fichier `icon.jpg` situé dans le répertoire `images`, présent deux répertoires au-dessus du répertoire courant.

Remarque

La description relative des chemins utilise la même syntaxe que la commande `cd` sous Unix.

Définition **Chemin absolu**

Un **chemin d'accès absolu** se décrit depuis la racine du serveur jusqu'à la ressource souhaitée.

Exemple

```
1 /images/icon.jpg
```

Le chemin d'accès absolu précédent pointe sur le fichier `icon.jpg` qui se trouve dans le répertoire `images`, présent à la racine du serveur.

Syntaxe **La création de répertoire**

PHP permet de créer un répertoire sur le système avec la fonction `mkdir()`.

Celle-ci prend en paramètres le chemin d'accès au répertoire à créer (qu'il soit relatif ou absolu), les droits à lui appliquer en notation octale, et un booléen spécifiant si la création récursive de répertoire est autorisée.

Exemple

```
1 <?php
2
3 mkdir('repertoire');
4 mkdir('exemple/repertoire/recursif', 0755, true);
```

Le code présenté va créer un dossier `repertoire` dans le dossier courant (ligne 1) et un dossier `exemple` contenant un dossier `repertoire`, dans lequel se trouve un dossier `recursif` (ligne 2).

Attention

L'affectation des droits par la fonction `mkdir()` sera toujours dépendante des droits autorisés de PHP.

Deux approches permettent de contourner ce fonctionnement :

- La première consiste à utiliser la fonction `umask()` en lui passant les limitations de droits à appliquer avant l'appel à `mkdir()`. En effet, cette fonction permet d'interdire certains niveaux d'accès aux ressources créées pour la durée d'exécution d'un script. Cela signifie qu'elle reprend sa valeur par défaut à la fin de chaque script PHP.
- La seconde approche est d'utiliser la fonction `chmod()` qui prend en paramètres le chemin d'accès à la ressource ciblée et les droits, en notation octale, à lui appliquer.

Exemple **Avec umask()**

```
1 <?php
2
3 $oldPermissions = umask(0); // Affecte la valeur actuelle des limitations de droits de fichier
  dans la variable $oldPermissions et autorise l'utilisation de n'importe quel niveau de
  permission grâce à la valeur 0.
4 mkdir('repertoireAvecUmask', 0777); // Crée un répertoire dans le répertoire courant avec des
  droits de lecture, écriture et exécution pour tous.
5 umask($oldPermissions ); // Réaffecte l'ancienne valeur  des limitations de droits de fichier
  pour les cas où le script continue.
```

Avec `chmod()`

```php
<?php

mkdir('repertoireAvecChmod', 0777); // Crée un répertoire dans le répertoire courant avec des droits de lecture, écriture et exécution pour tous si la valeur par défaut de restriction des droits de PHP l'autorise.
chmod('repertoireAvecChmod', 0777); // Change les droits du répertoire pour correspondre à ceux souhaités quelles que soient les restrictions de PHP.
```

La suppression de répertoire

Pour supprimer un répertoire, il est possible d'utiliser la fonction `rmdir()`, qui prend en premier paramètre le chemin vers le répertoire. Cela ne fonctionnera que si le répertoire est vide et si vous avez les bons droits.

Exemple

```php
<?php

rmdir('exemple/repertoire/recursif');
```

Parcourir un répertoire

Le parcours de répertoire en PHP se fait avec la fonction `scandir()` qui prend en paramètres le chemin du répertoire à parcourir, et un entier précisant l'ordre de tri des éléments retournés.

Cette fonction retourne ses résultats sous la forme d'un tableau, qu'il faudra ensuite parcourir pour effectuer des opérations sur le contenu du répertoire cible.

Exemple

```php
<?php

mkdir('repertoire');
mkdir('exemple/repertoire/recursif', 0755, true);

$results = scandir ('./');
foreach($results as $value) {
  echo $value.' ';
}

echo PHP_EOL;

$results = scandir('exemple/repertoire');
foreach($results as $value) {
  echo $value.' ';
}
```

Le code présenté crée un dossier `repertoire` et une arborescence de dossiers à la racine du dossier courant, puis récupère le contenu du répertoire courant et affiche les valeurs de chaque élément obtenu.

Le répertoire courant contient donc un élément `exemple`, une ressources `main.php` et un élément `repertoire`.

Un parcours du contenu du dossier `repertoire` contenu dans `exemple` montre ensuite qu'il contient un élément `recursif`.

```
. .. exemple main.php repertoire
. .. recursif
```

Attention

Au sein des systèmes Unix, chaque répertoire contient une référence vers lui-même, matérialisée par un `.`, et une référence vers son répertoire parent matérialisée par `..`.

Complément

Lors du parcours des éléments d'un répertoire, la fonction `is_dir()` peut être appelée afin de déterminer si l'élément en cours est un répertoire, et ainsi prévoir des traitements spécifiques pour chaque type d'éléments trouvés.

Exemple

```php
mkdir('repertoire');
mkdir('exemple/repertoire/recursif', 0755, true);

$results = scandir ('./');
foreach($results as $value) {
  if (is_dir($value)) {
    echo $value." est un répertoire \n";
  } else {
    echo $value." n'est pas un répertoire \n";
  }
}
```

```
. est un répertoire
.. est un répertoire
exemple est un répertoire
main.php n'est pas un répertoire
repertoire est un répertoire
```

Syntaxe **À retenir**

PHP offre des fonctions permettant la gestion de répertoires, ainsi que de leurs droits grâce aux fonctions :

- `mkdir()` pour la création d'un répertoire,
- `rmdir()` pour la suppression d'un répertoire,
- `is_dir()` pour déterminer s'il s'agit d'un répertoire ou non,
- `scandir()` pour récupérer le contenu d'un dossier.

III. Exercice : Appliquez la notion

Commencez par créer différents répertoires là où vous exécuterez votre code. Créez par exemple les répertoires "eric" "mathilde" et "manon".

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[1]

1 https://repl.it/

Question

[solution n°1 p.21]

En partant des données suivantes, écrivez un code qui créera un répertoire pour chaque utilisateur dans le répertoire courant de votre script et supprimera les répertoires inutiles.

Utilisateurs :

```php
$users = ['mathilde', 'nicolas', 'frank'];
```

Arborescence actuelle :

```
.
├── eric
├── mathilde
└── manon
```

Arborescence attendue :

```
.
├── frank
├── mathilde
└── nicolas
```

Attention, à ce jour, sous Replit, il semblerait que la suppression d'un dossier nécessite que l'interface soit rafraîchie pour être vérifiée.

IV. Manipuler des fichiers

Objectif

- Apprendre à ouvrir et fermer un fichier

Mise en situation

Avant de lire et récupérer des informations depuis un fichier présent sur le serveur, il est nécessaire de l'ouvrir.

Ce chapitre abordera la notion de **ressources en PHP**, avant de présenter les fonctions nécessaires à l'ouverture puis à la fermeture d'un fichier.

Les ressources en PHP

En PHP, on appelle ressource une variable contenant une référence vers une ressource externe. Celle-ci peut être de différents types, comme une référence vers une connexion à une base de données ou une référence vers un fichier. La liste des types de ressources peut être consultée ici : https://www.php.net[1].

Une référence vers une ressource est toujours créée lors de l'ouverture de la ressource externe ciblée. Afin de s'assurer de la bonne libération de la mémoire, il est nécessaire de s'assurer de sa fermeture lorsque les traitements voulus sont terminés. Si cela n'est pas fait, PHP gardera ouvert un flux référençant la ressource donnée.

Ouvrir un fichier

Lors du développement de fonctionnalités de gestion de fichiers locaux, il est rare d'en connaître à l'avance le chemin d'accès exact.

En revanche, il est courant de devoir parcourir une arborescence donnée et d'en explorer les fichiers de manière dynamique.

Pour cela, il est nécessaire d'utiliser la fonction `is_file()`, qui prend en paramètre le chemin de la ressource externe et retourne `true` s'il s'agit d'un fichier.

---

1 https://www.php.net/manual/fr/resource.php

On peut ensuite utiliser la fonction `fopen()`, qui prend en paramètres le chemin du fichier souhaité et le mode d'ouverture voulu, afin d'obtenir une ressource PHP permettant l'accès au fichier.

Exemple

```php
<?php

var_dump(is_file('fichier.txt'));
$result = fopen('fichier.txt', 'r');
var_dump($result);
```

Le code présenté ouvre le fichier `fichier.txt` présent dans le répertoire courant et affiche le résultat obtenu, qui est une ressource PHP de type `stream`.

```
true
resource(5) of type (stream)
```

Remarque **fopen()**

La fonction `fopen()` prend en deuxième paramètre une chaîne de caractères représentant le mode d'ouverture du fichier.

On peut ainsi, par exemple :

- ouvrir un fichier en lecture seule avec `'r'`, ce qui place le pointeur de la ressource au début du fichier.
- ouvrir un fichier en écriture seule avec `'w'`, ce qui efface le contenu du fichier, place le pointeur de la ressource au début du fichier et crée le fichier si celui-ci n'existe pas.
- créer et ouvrir un fichier en écriture seule avec `'x'`, mais génère une erreur si le fichier existe déjà. Si le fichier n'existe pas, `fopen()` tentera de le créer.
- ouvrir un fichier avec `'c'`, en écriture seul. Si le fichier n'existe pas, il sera crée.
- ouvrir un fichier en écriture seule avec `'a'`, ce qui place le pointeur de la ressource vers la fin du fichier.

Fermer un fichier

Une fois un fichier ouvert et les traitements souhaités réalisés, il est nécessaire de refermer le fichier pour libérer la référence vers celui-ci.

Pour cela, il est possible d'utiliser la fonction `fclose()` qui prend en paramètre la ressource PHP vers le fichier cible et qui renvoie un booléen représentant son succès.

Exemple

```php
<?php

$close = fclose($result);
var_dump($close, $result);
```

Ici, la ressource est fermée grâce à la fonction `fclose()`, et on observe que, suite au succès de la fermeture, la ressource n'a plus de pointeur valide.

```
bool(true)
resource(5) of type (Unknown)
```

Syntaxe **À retenir**

- La gestion des fichiers en PHP passe par l'ouverture et la fermeture de ressources pointant vers ces fichiers grâce aux fonctions `is_file()`, `fopen()` et `fclose()`.

V. Exercice : Appliquez la notion

Au sein de votre environnement de travail, créez les fichiers suivants :

- `fichier1.txt`
- `fichier2.txt`
- `fichier3.txt`

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[1]

Question

[solution n°2 p.21]

Écrivez les fonctions suivantes :

- la fonction `openFile()`, qui prendra en paramètre un nom de fichier, qui ouvrira le fichier en lecture seule et qui retournera une ressource PHP. Si le fichier n'existe pas, elle retournera `false`.
- la fonction `closeFile()`, qui prendra en paramètre une ressource PHP, qui la fermera et retournera un booléen indiquant si elle a réussi à fermer la ressource. Si le paramètre n'est pas une ressource, elle retournera `false`.

Vous pourrez vérifier votre code de la sorte :

```php
<?php
$ressource = openFile('fichier-existant');
// Affichera resource(5) of type (stream) par exemple
var_dump($ressource);

$isClosed = closeFile($ressource);
// Affichera bool(true)
var_dump($ressource);

$ressource = openFile('fichier-inexistant');
// Affichera bool(false)
var_dump($ressource);

$isClosed = closeFile($ressource);
// Affichera bool(false)
var_dump($ressource);
```

Indice :

La fonction PHP `is_resource()`[2] vous aidera à savoir si une variable est une ressource PHP.

Le fichier “fichier-existant” (nom donné pour l’exemple) doit déjà être créé avant le lancement du programme.

VI. Lire un fichier

Objectif

- Apprendre à lire dans un fichier

1 https://repl.it/
2 https://www.php.net/manual/fr/function.is-resource.php

Mise en situation

Lors de la réalisation de fonctionnalités de gestion de fichiers, il est courant de devoir en récupérer le contenu pour des traitements spécifiques ou de l'affichage.

Pour cela, il est nécessaire de savoir comment lire le contenu d'un fichier.

Lire un fichier ligne à ligne

Il est possible de lire le contenu d'un fichier, une ligne après l'autre, grâce à la fonction `fgets()`. Celle-ci prend en paramètres la ressource PHP pointant vers le fichier souhaité et, optionnellement, la taille en octets des données à lire.

Elle retourne le contenu de la ligne sur laquelle est positionné le pointeur de ressource, ou `false` s'il n'y a plus de données à lire.

Pour l'utiliser, il est donc nécessaire d'avoir au préalable ouvert un fichier.

Exemple

```php
<?php

$resource = fopen('fichier.txt', 'c+');

if ($resource) {
    while (($buffer = fgets($resource)) !== false) {
        echo $buffer;
    }

    fclose($resource);
}
```

Le code présenté crée, s'il n'existe pas, le fichier `fichier.txt`, puis l'ouvre en lecture-écriture.

Il vérifie ensuite que la ressource a bien été créée avant de boucler sur le contenu du fichier, en récupérant une nouvelle ligne à chaque passage tant que la récupération de données n'est pas terminée.

La ressource pointant vers le fichier est ensuite fermée.

```
Ligne 1
Ligne 2
Ligne 3
Ligne 4
```

La lecture par section

PHP permet aussi de récupérer tout ou partie d'un fichier grâce à la fonction `file_get_contents()`. Celle-ci prend en paramètres le chemin du fichier ciblé et accepte un indice de position de départ de la lecture des données, ainsi que la longueur des données à lire. Elle retourne les données obtenues sous la forme d'une chaîne de caractères, ou `false` en cas d'erreur.

Exemple

```php
<?php

$section = file_get_contents('fichier.txt');
var_dump($section);
```

Cet exemple affiche le contenu complet du fichier `fichier.txt`.

```
string(34) "Ligne 1
Ligne 2
Ligne 3
Ligne 4"
```

Remarque **file_get_contents()**

La méthode `file_get_contents()` se charge de réaliser l'ouverture et la fermeture du fichier.

Attention

L'utilisation de `file_get_contents()` au sein d'une boucle peut être consommateur de ressources, car elle ouvrira et fermera le fichier ciblé à chaque itération. De plus, elle peut se révéler moins performante si l'entièreté du contenu du fichier est chargée à chaque fois.

Syntaxe **À retenir**

- La lecture des données contenues dans un fichier peut se faire ligne par ligne avec la fonction `fgets()` ou par section avec la fonction `file_get_contents()`.

VII. Exercice : Appliquez la notion

Sur votre environnement de travail, créez un fichier texte contenant les éléments suivants :

```
Mathilde Dubois
Eric Blanchard
Manon Dupont
```

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[1]

Question

[solution n°3 p.21]

Écrivez la fonction `getUsersFromFile()`, qui prendra en paramètre un nom de fichier, ouvrira le fichier, le lira et renverra un tableau contenant les utilisateurs qui y sont stockés. Prenez garde à vérifier que le fichier existe et que la fonction `fopen()` renvoie bien une ressource et, en cas de problème, qu'elle retournera `null`.

Vous ferez en sorte que la méthode d'ouverture du fichier soit la plus optimisée possible.

Vous pourrez vérifier que tout est en ordre en effectuant, par exemple, l'appel de fonction suivant :

```php
<?php

var_dump(getUsersFromFile('fichier.txt'));

/*
array(3) {
  [0]=>
  string(15) "Mathilde Dubois"
  [1]=>
  string(14) "Eric Blanchard"
```

1 https://repl.it/

```
  [2]=>
  string(12) "Manon Dupont"
*/
```

Indice :

La fonction `trim()`[1] vous permettra de vous débarrasser des retours à la ligne en trop en début et en fin de chaîne de caractères.

VIII. Écrire dans un fichier

Objectif

- Apprendre à écrire dans un fichier

Mise en situation

Lors de la réalisation de fonctionnalités de gestion de fichiers, il est courant de devoir écrire du contenu dans un fichier local. Nous allons ici montrer comment PHP permet l'écriture dans un fichier.

Écriture dans un fichier

L'écriture dans un fichier en PHP peut se faire avec la fonction `fwrite()`. Celle-ci prend en paramètres la ressource pointant vers le fichier cible, ainsi que la chaîne de caractères à écrire, et peut aussi recevoir la longueur d'écriture souhaitée.

Exemple

```php
$resource = fopen('fichier.txt', 'w');

fwrite($resource, 'Hello');
fwrite($resource, 'world');
fwrite($resource, '!');

fclose($resource);

var_dump(file_get_contents('fichier.txt'));
```

L'exemple présenté ouvre en écriture le fichier `fichier.txt` et écrit le texte '*Helloworld!*'.

```
string(11) "Helloworld!"
```

Remarque **fwrite()**

La fonction `fwrite()` écrit les données à partir de la position actuelle du pointeur et positionne celui-ci à la fin des données qu'elle a écrites.

file_put_contents()

Il est aussi possible d'écrire dans un fichier en utilisant la fonction `file_put_contents()`.

Celle-ci prend en paramètres le chemin du fichier ciblé, les données à écrire sous la forme d'une chaîne de caractères, d'un tableau ou d'une ressource.

De plus, il est possible de lui spécifier si l'ajout du contenu doit se faire à la suite des données existantes, ou si ces dernières doivent être écrasées (ce qui est le comportement par défaut).

1 https://www.php.net/manual/fr/function.trim.php

Exemple

```php
file_put_contents('fichier.txt', 'Hello world !'.PHP_EOL);
file_put_contents('fichier.txt', ['Texte', 'supplémentaire'], FILE_APPEND);

var_dump(file_get_contents('fichier.txt'));
```

L'exemple illustre l'utilisation de la fonction `file_put_content()` en écrivant le texte *Hello world !* et en lui ajoutant le contenu d'un tableau pour écrire *Texte supplémentaire*.

```
string(34) "Hello world !
Textesupplémentaire"
```

Remarque

La méthode `file_put_contents()` se charge de réaliser l'ouverture et la fermeture du fichier.

Attention

L'utilisation de `file_put_contents()` au sein d'une boucle peut être consommateur de ressources, car elle ouvrira et fermera le fichier ciblé à chaque exécution.

Syntaxe **À retenir**

- L'écriture des données dans un fichier peut se faire grâce à la fonction `fwrite()` ou à la fonction `file_put_contents()`.

IX. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°4 p.22]

Mettez en place une fonction qui prendra en paramètres un nom de fichier, ainsi qu'un tableau d'utilisateurs contenant leur nom et prénom.

Cette fonction ouvrira le fichier (en prenant soin de le remettre à 0), écrira les données et retournera `true`. Prenez garde que la fonction `fopen()` renvoie bien une ressource et, en cas de problème, qu'elle retournera `false`.

Vous ferez en sorte que la méthode d'ouverture du fichier soit la plus optimisée possible.

Tableau d'utilisateurs :

```php
$users = [
    ['Mathilde', 'Dubois'],
    ['Eric', 'Blanchard'],
    ['Manon', 'Dupont'],
];
```

1 https://repl.it/

Résultat attendu :

```
Mathilde Dubois
Eric Blanchard
Manon Dupont

```

Indice :

La fonction `implode()`[1] vous permet de concaténer un tableau avec une glue.

X. Lire les métadonnées

Objectif

- Apprendre à lire les métadonnées d'un fichier

Mise en situation

Il est courant de ne devoir réaliser un traitement sur un fichier qu'en fonction de ses propriétés.

Pour cela, il est nécessaire de savoir ce que sont les métadonnées d'un fichier, et comment récupérer certaines de ces informations.

Remarque **Les métadonnées d'un fichier**

Les métadonnées d'un fichier correspondent à l'ensemble des informations d'un fichier qui ne sont pas contenues dans les données. Ce sont, par exemple, sa taille, sa date de dernière modification, ses permissions, etc.

PHP propose différentes fonctions pour accéder à ces informations.

Les informations de date

- La fonction `filectime()` permet de lire la dernière date de modification des propriétés d'un fichier. Elle prend en paramètre le chemin d'accès au fichier et renvoie une date sous la forme d'un timestamp Unix.
- La fonction `filemtime()` permet de lire la date de dernière modification des données d'un fichier. Elle prend en paramètre le chemin d'accès au fichier et renvoie une date sous la forme d'un timestamp Unix.
- La fonction `fileatime()` permet de lire la date du dernier accès à un fichier. Elle prend en paramètre le chemin d'accès au fichier et renvoie une date sous la forme d'un timestamp Unix.

Exemple

```php
<?php

var_dump(date('c', filectime('fichier.txt')));
var_dump(date('c', filemtime('fichier.txt')));
var_dump(date('c', fileatime('fichier.txt')));
```

```
<?php

string(25) "2020-04-08T09:50:58+00:00"
string(25) "2020-04-08T09:50:58+00:00"
string(25) "2020-04-08T09:50:58+00:00"
```

1 https://www.php.net/manual/fr/function.implode.php

Les informations de permissions

- La fonction `fileowner()` permet de récupérer le propriétaire d'un fichier. Elle prend en paramètre le chemin d'accès au fichier et renvoie l'identifiant du propriétaire du fichier.
- La fonction `filegroup()` permet de lire le groupe d'un fichier. Elle prend en paramètre le chemin d'accès au fichier et renvoie l'identifiant du groupe auquel le fichier appartient.
- La fonction `fileperms()` permet d'obtenir les droits d'accès à un fichier. Elle prend en paramètre le chemin d'accès au fichier et renvoie les permissions du fichier en valeur octale.

Ces informations peuvent nécessiter ensuite d'utiliser des fonctions systèmes permettant de récupérer des valeurs utilisables, comme `posix_getpwuid()` pour obtenir les information liées au propriétaire, ou `posix_getgrgid()` pour les informations liées au groupe.

Exemple

```php
<?php

var_dump('Propriétaire : ', posix_getpwuid(fileowner('fichier.txt')));
var_dump('Groupe : ', posix_getgrgid(filegroup('fichier.txt')));
var_dump('Droits : ', substr(sprintf('%o', fileperms('fichier.txt')), -4)); // %o permet d'afficher la notation octale
```

```
string(15) "Propriétaire : "
array(7) {
  ["name"]=>
  string(6) "runner"
  ["passwd"]
  string(1) "x"
  ["uid"]=>
  int(1000)
  ["gid"]=>
  int(1000)
  ["gecos"]=>
  string(0) ""
  ["dir"]=>
  string(12) "/home/runner"
  ["shell"]=>
  string(9) "/bin/bash"
}
string (9) "Groupe : "
array(4) {
  ["name"]=>
  string(6) "runner"
  ["passwd"]=>
  string(1) "x"
  ["members"]=>
  array(0) {
  }
  ["gid"]=>
  int(1000)
}
string(9) "Droits : "
string(4) "0644"
```

Les informations de taille

La fonction `filesize()` permet de récupérer le taille d'un fichier. Elle prend en paramètre le chemin d'accès au fichier et retourne sa taille en octets.

Exemple

```php
<?php

var_dump('Taille : ', filesize('fichier.txt').' octets');
```

```
string(9) "Taille : "
string(9) "54 octets"
```

Syntaxe **À retenir**

- Les métadonnées d'un fichier représentent les propriétés qui lui sont liées. Elles peuvent être obtenues grâce à des fonctions spécifiques de PHP, telles que `filectime()`, `filegroup()`, `fileperms()`, `fileowner()`,`filegroup()` ou `filesize()`.

XI. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[1]

Question

[solution n°5 p.22]

Écrivez une fonction qui listera les fichiers et répertoires d'un répertoire passé en paramètre, et qui affichera pour chacun sa date de création et de dernière modification. Comme ceci :

```
.env a été créé le 27/03/2020 à 14:06:39 et modifié le 27/03/2020 à 14:06:39
bin a été créé le 23/03/2020 à 14:52:12 et modifié le 23/03/2020 à 14:52:12
composer.json a été créé le 07/04/2020 à 09:03:40 et modifié le 07/04/2020 à 09:03:40
composer.lock a été créé le 07/04/2020 à 09:03:42 et modifié le 07/04/2020 à 09:03:42
config a été créé le 07/04/2020 à 09:02:21 et modifié le 07/04/2020 à 09:02:21
public a été créé le 08/04/2020 à 12:56:28 et modifié le 08/04/2020 à 12:56:28
src a été créé le 07/04/2020 à 09:02:39 et modifié le 07/04/2020 à 09:02:39
symfony.lock a été créé le 07/04/2020 à 09:03:42 et modifié le 07/04/2020 à 09:03:42
var a été créé le 23/03/2020 à 14:52:12 et modifié le 23/03/2020 à 14:52:12
vendor a été créé le 07/04/2020 à 09:03:42 et modifié le 07/04/2020 à 09:03:46
```

Vous pourrez vérifier que tout est en ordre de telle sorte :

```php
<?php
// on liste les fichiers du répertoire courant
listFilesAndDirectories('.');
```

Indice :

La fonction `realpath()`[2] permet de retourner le chemin absolu d'un fichier ou d'un dossier.

1 https://repl.it/
2 https://www.php.net/manual/fr/function.realpath.php

XII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°6 p.23]

Exercice

Lesquels de ces chemins sont des chemins absolus ?

- [ ] src/Controller/UserController.php
- [ ] /src/Controller/UserController.php
- [ ] ./src/Controller/UserController.php
- [ ] /app/../src/Controller/UserController.php

Exercice

Quelle fonction permet de créer un répertoire ?

- [ ] `mkdir('repertoire')`
- [ ] `mkdir(0777, 'repertoire')`
- [ ] `mkdir('repertoire', 0777)`
- [ ] `rmdir('repertoire')`

Exercice

Quelle fonction permet de récupérer le contenu d'un répertoire ?

Exercice

Quelle fonction permet de vérifier qu'un élément est un fichier ?

Exercice

Quel code permet d'ouvrir un fichier en lecture seule ?

- [ ] fopen('fichier.txt', 'w')
- [ ] fopen('fichier.txt', 'r')
- [ ] fopen('r', 'fichier.txt')
- [ ] fopen('w', 'fichier.txt')

Exercice

Que renvoie la fonction `fopen()` ?

- [ ] Une chaîne de caractères
- [ ] Une ressource
- [ ] Un entier
- [ ] Un booléen

Exercice

Quelle syntaxe permet d'itérer sur le contenu d'une ressource, ligne par ligne ?

- ☐ while (!($buffer = fgets($resource)))
- ☐ while (($buffer = fgets($resource)) != false)
- ☐ while (($buffer = fgets($resource)) !== false)

Exercice

Comment ferme-t-on un fichier ouvert avec la méthode `fopen()` ?

- ☐ close($resource)
- ☐ close('fichier.txt')
- ☐ fclose($resource)
- ☐ fclose('fichier.txt')

Exercice

Quelle fonction va permettre d'écrire dans un fichier en gérant implicitement son ouverture son écriture et sa fermeture ?

Exercice

Quelles informations ne font pas partie des métadonnées d'un fichier ?

- ☐ Le contenu
- ☐ La date de dernière modification
- ☐ La taille
- ☐ Les permissions

B. Exercice : Défi

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°7 p.24]

Écrivez une fonction qui permettra d'initialiser une arborescence de fichiers selon les critères suivants :

- La fonction prendra en paramètres le répertoire racine et la liste des utilisateurs.
- Pour chaque utilisateur, elle créera les dossiers décrits ci-après, en regroupant les utilisateurs qui ont le même nom.
- Ajoutez à la racine du dossier de chaque utilisateur un fichier `welcome.txt` contenant le message `Bienvenue {prenom} {nom}`. Si le fichier existe déjà, il ne doit pas être modifié.
- La fonction renverra un tableau avec, pour chaque utilisateur, le chemin vers le fichier `welcome.txt` et sa taille, comme décrit ci-après.

1 https://repl.it/

Utilisateurs :

```php
$users = [
    ['firstname' => 'Mathilde', 'lastname' => 'Dubois'],
    ['firstname' => 'Eric', 'lastname' => 'Blanchard'],
    ['firstname' => 'Manon', 'lastname' => 'Dubois'],
];
```

Retour attendu :

```php
$users = [
    ['firstname' => 'Mathilde', 'lastname' => 'Dubois', 'file' => ['path' => '/app/public/dubois/mathilde/welcome.txt', 'size' => 25]],
    ['firstname' => 'Eric', 'lastname' => 'Blanchard', 'file' => ['path' => '/app/public/blanchard/eric/welcome.txt', 'size' => 24]],
    ['firstname' => 'Manon', 'lastname' => 'Dubois', 'file' => ['path' => '/app/public/dubois/manon/welcome.txt', 'size' => 22]],
];
```

Arborescence attendue :

```
.
|-- blanchard
|   `-- eric
|       `-- welcome.txt
`-- dubois
    |-- manon
    |   `-- welcome.txt
    `-- mathilde
        `-- welcome.txt

```

Indice :

Notre programme pourrait être exécuté de la sorte : `initUserFolders('.', $users);`.

Solutions des exercices

p. 7 **Solution n°1**

```php
<?php
$users = ['bill', 'bob', 'joe'];
$users = ['mathilde', 'nicolas', 'frank'];

foreach ($users as $user) {
    if (!is_dir($user)) {
        mkdir(strtolower($user));
    }
}

$directories = scandir('.');

foreach ($directories as $directory) {
    if ($directory !== '.' && $directory !== '..' && is_dir($directory) && !in_array($directory, $users)) {
        rmdir($directory);
    }
}
```

p. 9 **Solution n°2**

```php
<?php

function openFile (string $myFile) {
return is_file($myFile) ? fopen($myFile, 'r') : false;
}

function closeFile($myResource) {
return is_resource($myResource) ? fclose($myResource) : false;
}

$ressource = openFile('fichier-existant');
// Affichera resource(5) of type (stream) par exemple
var_dump($ressource);

$isClosed = closeFile($ressource);
// Affichera bool(true)
var_dump($isClosed);

$ressource = openFile('fichier-inexistant');
// Affichera bool(false)
var_dump($ressource);

$isClosed = closeFile($ressource);
// Affichera bool(false)
var_dump($isClosed);
?>
```

p. 11 **Solution n°3**

```php
<?php

function getUsersFromFile(string $file): ?array {
    $users = [];

    if (!is_file($file)) {
        return null;
    }

    $resource = fopen($file, 'r');

    if (!$resource) {
        return null;
    }

    while (($user = fgets($resource)) !== false) {
        $users[] = trim($user);
    }

    fclose($resource);

    return $users;
}
```

p. 13 **Solution n°4**

```php
<?php

function writeUsersInFile(string $file, array $users): bool {
    $resource = fopen($file, 'w');

    if (!$resource) {
        return false;
    }

    foreach ($users as $user) {
        fwrite($resource, implode(' ', $user).PHP_EOL);
    }

    fclose($resource);

    return true;
}

$users = [
    ['Mathilde', 'Dubois'],
    ['Eric', 'Blanchard'],
    ['Manon', 'Dupont'],
];

writeUsersInFile('ecrire-fichier.txt', $users);
```

p. 16 **Solution n°5**

```php
function listFilesAndDirectories(string $path): void {
    $path = realpath($path).DIRECTORY_SEPARATOR;
    $elements = scandir($path);

    foreach ($elements as $element) {
        echo $element;
        echo ' a été créé le '.date('d/m/Y à H:i:s', filectime($path.$element));
        echo ' et modifié le '.date('d/m/Y à H:i:s', filemtime($path.$element));
        echo PHP_EOL;
    }
}
```

La ligne 2, “$path = realpath($path).DIRECTORY_SEPARATOR;” permet de retourner le chemin absolu d’un fichier ou d’un dossier. DIRECTORY_SEPARATOR est une constante pré-définie qui permet de changer les “/” en fonction du système d’exploitation. “/” sur Linux et “\” sur Windows.

Exercice p. 17 Solution n°6

Exercice

Lesquels de ces chemins sont des chemins absolus ?

- ☐ src/Controller/UserController.php
- ☑ /src/Controller/UserController.php
- ☐ ./src/Controller/UserController.php
- ☑ /app/../src/Controller/UserController.php

Tous les chemins commençant par un **/** sont des chemins absolus, les autres chemins ne le sont pas.

Exercice

Quelle fonction permet de créer un répertoire ?

- ☑ `mkdir('repertoire')`
- ☐ `mkdir(0777, 'repertoire')`
- ☑ `mkdir('repertoire', 0777)`
- ☐ `rmdir('repertoire')`

Exercice

Quelle fonction permet de récupérer le contenu d'un répertoire ?

scandir

Exercice

Quelle fonction permet de vérifier qu'un élément est un fichier ?

is_file

Exercice

Quel code permet d'ouvrir un fichier en lecture seule ?

- [ ] fopen('fichier.txt', 'w')
- [x] fopen('fichier.txt', 'r')
- [ ] fopen('r', 'fichier.txt')
- [ ] fopen('w', 'fichier.txt')

Exercice

Que renvoie la fonction `fopen()` ?

- [ ] Une chaîne de caractères
- [x] Une ressource
- [ ] Un entier
- [x] Un booléen

  Dans le cas d'une erreur, la fonction `fopen()` renverra `false`.

Exercice

Quelle syntaxe permet d'itérer sur le contenu d'une ressource, ligne par ligne ?

- [ ] while (!($buffer = fgets($resource)))
- [x] while (($buffer = fgets($resource)) != false)
- [x] while (($buffer = fgets($resource)) !== false)

Exercice

Comment ferme-t-on un fichier ouvert avec la méthode `fopen()` ?

- [ ] close($resource)
- [ ] close('fichier.txt')
- [x] fclose($resource)
- [ ] fclose('fichier.txt')

Exercice

Quelle fonction va permettre d'écrire dans un fichier en gérant implicitement son ouverture son écriture et sa fermeture ?

file_put_contents

Exercice

Quelles informations ne font pas partie des métadonnées d'un fichier ?

- [x] Le contenu
- [ ] La date de dernière modification
- [ ] La taille
- [ ] Les permissions

p. 18 **Solution n°7**

```php
<?php

function initUserFolders(string $rootFolder, array $users): array
{
    $rootFolder = realpath($rootFolder) . DIRECTORY_SEPARATOR;

    foreach ($users as $index => $user) {
        $userFolder = $rootFolder . strtolower($user['lastname']) . DIRECTORY_SEPARATOR . strtolower($user['firstname']) . DIRECTORY_SEPARATOR;
        $userFile = $userFolder . 'welcome.txt';

        if (!is_dir($userFolder)) {
            mkdir($userFolder, 0777, true);
        }

        if (!is_file($userFile)) {
            $resource = fopen($userFile, 'w');
            fwrite($resource, 'Bienvenue ' . implode(' ', $user));
            fclose($resource);
        }

        $users[$index]['file'] = [
            'path' => $userFile,
            'size' => filesize($userFile),
        ];
    }

    return $users;
}

$users = [
    ['firstname' => 'Mathilde', 'lastname' => 'Dubois'],
    ['firstname' => 'Eric', 'lastname' => 'Blanchard'],
    ['firstname' => 'Manon', 'lastname' => 'Dubois'],
];

initUserFolders('.', $users);
```