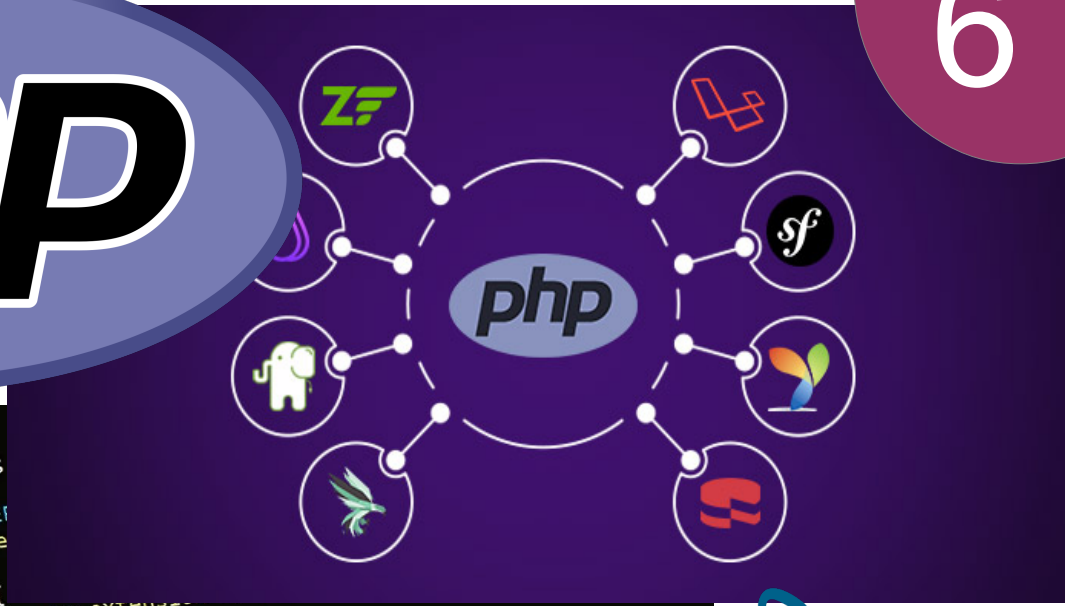
php

6




```
 * @var boolean
 */
define('PSI_INTERNAL_XML', false);

if (version_compare("5.2", PHP_VE
    die("PHP 5.2 or greater is re

}
if (!extension_loaded("pcre")) {
    die("phpSysInfo requires the pcre extens
        properly.");

}

require_once APP_ROOT.'/includes/autoloader.inc.php';

// Load configuration
require_once APP_ROOT.'/config.php';

if (!defined('PSI_CONFIG_FILE') || !defined('PSI_DEBUG')) {
    $tpl = new Template("/templates/html/error_config.html");
    echo $tpl->fetch();
    die();
```

MySQL®

Messmer Lucie

Gestion de fichiers

Le PHP va nous permettre de faire de nombreuses manipulations sur les fichiers. Des opérations classiques comme la lecture, l'écriture, la suppression, la copie, le déplacement, le renommage mais d'autres plus complexes permettant de télécharger ou de créer des fichiers spécifiques comme du PDF, du CSV, du XML, des formats compressés, des images, etc.
Certaines de ces operations (lecture, écriture) vont nécessiter un droit droit sur ces fichiers. C'est le cas sur les systemes de type Linux (la majorite des serveurs d'hebergement). Ces systèmes étant multi-utilisateur, il faut pouvoir définir des accès par sécurité et cela s'exprime par les **CHMOD**.
La plupart des logiciels FTP vous permettent de les modifier. Le CHMOD 777 sur un dossier permet par exemple la lecture et l'écriture pour tous les utilisateurs.

Renommer et déplacer

Une même fonction permet de faire les deux opérations, éventuellement simultanèment.
NB : fonctionne également sur les dossiers

Syntaxe

```
rename(ancien_nom, nouveau_nom)
```

exemple :

***Attention** : si on déplace un fichier vers un emplacement contenant un fichier de même nom, ce dernier sera écraser et remplacer.*

```
//  Renommer un fichier
rename('temp.doc', 'final.doc');
// Renommer et déplacer un fichier
rename('tmp/temp.doc', 'doc/final.doc');
```

Copier

Syntaxe

copy(ancien_nom, nouveau_nom)

exemple :

```
// Copie d'un fichier pour archivage
$fichier = 'texte_1.doc';
$sauvegarde = 'archives/texte_1.doc';
if (!copy($fichier, $sauvegarde)) {
	echo "La sauvegarde de " . $fichier . " a echoué...";
} else {
	echo "Sauvegarde du fichier " . $fichier . " réussie" ;
}
```

Supprimer

Syntaxe :

unlink(fichier);

rmdir(dossier);

exemple :

NB : *ces fonctions renvoient un booleen exprimant la réussite ou l'échec de l'opération.*

```
unlink('temp/texte_1.doc');
rmdir('temp');
```

Création d'un dossier

Syntaxe

```
mkdir(dossier, chmod)
```

NB : Il faut un zéro devant le chmod car il s'exprime en octal dans cette fonction.

Le chmod 755 donne les droits en lecture et éxécution à tout le monde mais les droits en écriture seulement au propriétaire.

exemple :

```
// creation d'un dossier (par defaut sera en chmod 777)
mkdir('temp');
// creation d'un dossier en chmod 755
mkdir('temp', 0755);
```

Création d'un fichier

On ne peut créer un fichier sans lui donner de contenu.

C'est donc en intervenant sur le contenu d'un fichier (avec une opération d'écriture) que celui-ci sera éventuellement créé s'il n'existe pas.

Pour toute manipulation des données d'un fichier, il y a trois étapes nécessaires :

1. Ouverture du fichier
2. Opérations
3. Fermeture du fichier

Ouverture

Syntaxe :

```
$pointeurFichier = fopen(fichier, mode)
```

Cette fonction génère une ressource que l'on va ranger dans une variable (appelée ''pointeur de fichier'') afin de pouvoir procéder aux manipulations à venir. Le mode permet de specifier un type d'ouverture.

'r' → Lecture seule, pointeur en début de fichier.
'r+' → Lecture et écriture, pointeur en début de fichier.
'w' → Ecriture seule, pointeur en début de fichier et vidage du fichier.
'w+ → Lecture et écriture, pointeur en début de fichier et vidage du fichier.
'a' → Ecriture seule, pointeur en fin de fichier.
'a+ → Lecture et écriture, pointeur en fin de fichier.

Fermeture

Syntaxe

```
fclose($pointeurFichier)
```

Grâce au pointeur de fichier défini a l'ouverture, nous savons quelle ressource refermer.

Ecriture dans un fichier

Syntaxe :

```
fwrite($pointeurFichier, texte)
```

Mais avant d'écrire à l'intérieur d'un fichier, il peut être judicieux de vérifier son existence et la possibilité d'écriture dans celui-ci.
Ces deux fonctions renvoient un boolean :

```
file_exists(fichier);   is_writable(fichier);
```

fonction FPUTS

Syntaxe :

`fputs(pointeur,string)`

Le contenu de la string est placé dans le fichier (en mode binaire). Si le mode r+ est choisi, la string sera insérée en début de fichier, écrasant éventuellement les données présentes.
Si le mode w ou w+ est choisi, le fichier sera ''vide'' et la string insérée.
Enfin si le mode a ou a+ est choisi, la string sera ajoutée en fin de fichier.

exemple :

NB: I*l est possible de déplacer notre pointeur dans le fichier. Si vous avez ouvert celui-ci en a ou a+, il ne sera possible d'écrire qu'en fin de fichier.*

```
// initialisation
$fichier = "fichier.txt";
// test d'écriture
if (is_writable($fichier)) {
// ouverture du fichier en ecriture
$pointeur = fopen($fichier, 'a');
// insertion d'une chaîne en fin de fichier
fputs($pointeur, "Insertion en fin de fichier !\n");
// fermeture du fichier
fclose($pointeur);
} else {
// problème d'ecriture
echo "Impossible d'écrire dans ce fichier.";
}
```

Fonction FGETS : lecture dans un fichier

Syntaxe

fgets($pointeur)

Cette fonction récupère le contenu de la ligne courante sur laquelle se trouve le pointeur du fichier.

Il est tout à fait possible de faire une boucle qui permettra de récuperer chaque ligne à chaque passage.

exemple :

```
$fichier = "fichier.txt";
// ouverture du fichier en lecture, pointeur au début
$pointeur = fopen($fichier, 'r');
// boucle parcourant et affichant chaque ligne du fichier
while ($ligne = fgets($pointeur)) {
	echo $ligne."<br>";
}
// fermeture du fichier
fclose($pointeur);
```

fonction FREAD

Syntaxe :

fread($pointeur, longueur)

Cette fonction va récupérer le contenu du fichier sans tenir compte des sauts de ligne.

C'est le paramètre longueur qui lui donnera une limite de lecture (en octets).

Lorsqu'on veut lire l'intégralité du fichier, on récupère sa taille en octet avec la fonction **filesize()**.

```
$fichier = "fichier.txt";
// ouverture du fichier en lecture, pointeur au début
$pointeur = fopen($fichier, 'r');
// lecture de l'intégralité du fichier
$contenu = fread($pointeur,filesize($fichier));
// affichage du contenu
echo $contenu;
// fermeture du fichier
fclose($pointeur);
```

Se déplacer dans un fichier

Syntaxe :

`fseek($pointeur, position);`

Avec cette fonction, nous allons pouvoir déplacer notre pointeur à un endroit précis du fichier.
La position s'exprime en octet. Utiliser une position à 0 permettra donc de replacer le pointeur au début de fichier, tout comme la fonction rewind($pointeur).

```
$fichier = "fichier.txt";
// ouverture du fichier en lecture, pointeur au début
$pointeur = fopen($fichier, 'r');
// lecture de l'intégralité du fichier (notre pointeur sera donc en fin de fichier)
$ligne = fread($pointeur,filesize($fichier));
// repositionnement du pointeur en début de fichier
fseek($pointeur, 0);
// fermeture du fichier
fclose($pointeur);
```

Ouvrir et fermer un dossier

Tout comme pour les fichiers, nous pouvons placer un pointeur dans un dossier précis afin de le parcourir. Il nous faut là aussi respecter la même succession d'operations :

1. Ouverture du dossier
2. Opérations
3. Fermeture du dossier

Ouverture

Syntaxe :

`opendir(dossier)`

Ouvre le dossier spécifié et place un pointeur dessus.

Cette fonction retourne ce qu'on appelle une "ressource de dossier" (ou false en cas d'echec).

Fermeture

Syntaxe :

`closedir(ressource)`

Ferme le pointeur sur le dossier specifié.

Lire le contenu d'un dossier

Syntaxe :

`readdir(ressource)`

Permet de lire une entrée du dossier (celle où se trouve le pointeur).

Créer une boucle adaptée permettra de parcourir toutes les entrées.

exemple :

```
// ouverture du dossier
$pointeur = opendir("archives");
// lecture de l'intégralité du dossier
while($entree = readdir($pointeur)) {
// affichage de l'entrée
 echo $entree."<br>";
}
// fermeture du dossier
closedir($pointeur);
```

Nature des entrées d'un dossier

Les entrées affichées ne permettent pas toujours de savoir s'il s'agit de dossiers ou de fichiers.
Des fonctions dediées vont nous aider :

Syntaxe :

```
is_file(entrée); //fichier
is_dir(entrée) ; //dossier
```

Ces deux fonctions testent l'entrée et renvoient true ou false selon qu'il s'agit ou non d'un fichier ou d'un dossier.

NB : *La fonction filetype(entrée) peut être une alternative : elle renvoi une chaîne de caractères indiquant la nature de la ressource passée en paramètre. ('dir', 'file', 'unknow'...)*

Informations accessibles

Fonctions permettant de récupérer des informations intéressantes sur les ressources rencontrées :

`filemtime(fichier)`	→ dernière modification (timestamp)
`filesize(fichier)`	→ taille du fichier (en octets)
`disk_total_space(partition)`	→ espace total occupé (en octets)
`disk_free_space(partition)`	→ espace total libre (en octets)

exemple :

```
// affiche la date de dernière modification d'un fichier de manière lisible
echo date("d/m/Y H:i", filemtime("fichier.txt"));
// affiche la taille d'un fichier de manière lisible
echo round((filesize("fichier.txt")/1024),1);
```