A comprehensive guide to PHP file operations

- By Adebayo Adams
- on Mar 26, 2024

Everything you do when using computers is related to files and folders. The browser you're using to view this webpage is a file on your device, and this webpage is a file on this website's server, which means everything is a file one way or another. In this article, you will learn everything you need to know to build files and folder-related features in your PHP applications.

Prerequisites

To follow along with this article, you need to have a solid grasp of the basics of PHP. Knowledge of variables, functions, conditionals, etc., will come in handy.

However, every concept and code example covered in this article will be adequately explained to ensure everything is clear.

File operations in PHP

File handling is an essential part of any web application, and all programming languages provide a way to perform these tasks in your applications. PHP is no exception, so it gives an extensive list of functions you can use when working with files in your applications.

The following sections will explore these functions and what they allow you to do in your PHP applications.

Reading files in PHP

Reading files is the most common file-related task in programming. PHP provides different ways to do this in your applications, and we will be exploring them in the following sections.

Read the file into a string variable

PHP allows you to read files as strings with the *file_get_contents* function. Here's an example:

```php
$fileContents = file_get_contents('file.txt');

echo $fileContents;
```

The code above returns all the content in the *file.txt* as a string or a warning if the file is unavailable in the directory.

Read file line by line

You can easily read a file line by line in PHP by using the *fopen* function with a loop like this:

```php
if (file_exists('file.txt')) {
    $file = fopen('file.txt', 'r');
    while (($line = fgets($file)) !== false) {
        echo $line . '<br>';
    }
    fclose($file);
} else {
    echo 'File does not exist';
}
```

The code above checks if the file exists using the *file_exists* function and opens the *file.txt* file in read mode; it then uses a *while* loop and the *fgets* function to read and print out the file line by line. Finally, after completing the loop, the code uses the *fclose* function to close the file.

The code above will return the following based on my *file.txt* file:

```
This is a sample text file.
It contains some text for testing purposes.
You can add more text here.
```

Note: You should always check if a file exists before operating on it and close files after working on them. Doing these would help minimize the number of bugs you'll have to fix.

Let's explore the different file opening modes in the next section.

Understanding the file opening modes in PHP

The `fopen` function in PHP is used to open files and returns a file pointer/resource that can be used for various file operations. The function accepts a filename and an optional mode string that specifies the intended file operation and other options. The following is a list of common mode options for `fopen`.

`r` (Read)

Open the file for reading. The file pointer is placed at the beginning of the file. If the file does not exist, it will return `false`.

`r+` (Read/Write)

Open the file for both reading and writing. The file pointer is placed at the beginning of the file. If the file does not exist, it will return `false`.

`w` (Write)

Open the file for writing. If the file does not exist, it will be created. If the file exists, its contents will be truncated. The file pointer is placed at the beginning of the file.

`w+` (Read/Write)

Open the file for both reading and writing. If the file does not exist, it will be created. If the file exists, its contents will be truncated. The file pointer is placed at the beginning of the file.

`a` (Append)

Open the file for writing. If the file does not exist, it will be created. The file pointer is placed at the end of the file, allowing you to append data without overwriting existing content.

`a+` (Append/Read)

Open the file for reading and writing. If the file does not exist, it will be created. The file pointer is placed at the end of the file.

`x` (Exclusive Create)

Open the file for writing only if it does not already exist. If the file exists, `fopen` will return `false`.

`x+` (Exclusive Create/Read)

Open the file for reading and writing only if it does not already exist. If the file exists, `fopen` will return `false`.

`c` (Open for Writing)

Open the file for writing. If the file does not exist, it will be created. Unlike *w*, it does not truncate the file. The file pointer is placed at the beginning of the file.

c+ (Open for Reading/Writing)

Open the file for reading and writing. If the file does not exist, it will be created. Unlike *w+*, it does not truncate the file. The file pointer is placed at the beginning of the file.

e (Ephemeral)

Open the file in "ephemeral" mode. It behaves like *w* but does not affect the file modification time. It was introduced in PHP 7.4.

b (Binary mode)

Append *b* to any of the above modes (e.g., *rb*, *wb+*) to indicate binary mode, which is used when working with binary files.

t (Text mode)

Append *t* to any of the above modes (e.g., *rt*, *w+t*) to indicate text mode, which is the default mode used for text files on most platforms.

Note: The behavior of *fopen* may vary slightly between different operating systems. Also, some modes may have platform-specific nuances or restrictions. Always check the PHP documentation and consider error handling when using *fopen* to ensure your code behaves as expected.

Reading `.csv` files in PHP

PHP provides a *`fgetcsv`* function for handling .csv files in your PHP applications. You can read and parse .csv files like this:

```
if (file_exists('file.csv')) {
    $csvFile = fopen('file.csv', 'r');
    while (($data = fgetcsv($csvFile)) !== false) {
        echo '<pre>';
        print_r($data);
        echo '</pre>';
    }
    fclose($csvFile);
} else {
    echo 'File does not exist';
}
```

The code above does the same thing as the previous one but prints the CSV data as an associative array. The code will return the following based on my *`file.csv`* file:

```
Array
(
    [0] => Name
    [1] => Age
    [2] => Email
)
Array
(
    [0] => John Doe
    [1] => 30
    [2] => john@example.com
)
Array
(
    [0] => Jane Smith
    [1] => 25
    [2] => jane@example.com
)
Array
(
    [0] => Bob Johnson
    [1] => 35
    [2] => bob@example.com
)
```

Reading `.json` files in PHP

PHP enables you to read and decode *`.json`* files using the *`file_get_contents`* and *`json_decode`* functions like this:

```
if (file_exists('file.json')) {
    $jsonString = file_get_contents('file.json');
    $jsonData = json_decode($jsonString, true);
    echo '<pre>';
    print_r($jsonData);
    echo '</pre>';
} else {
    echo 'File does not exist';
}
```

The code above will print out the content of the *`file.json`* file as an associative array:

```
Array
(
    [0] => Array
        (
            [name] => John Doe
            [age] => 30
            [email] => john@example.com
            [address] => Array
                (
                    [street] => 123 Main St
                    [city] => Anytown
                    [state] => CA
                )
            [hobbies] => Array
                (
                    [0] => Reading
                    [1] => Gardening
                    [2] => Cooking
                )
        )
    [1] => Array
        (
            [name] => Jane Doe
            [age] => 25
            [email] => jane@example.com
            [address] => Array
                (
                    [street] => 123 Alao St
                    [city] => Surulere
                    [state] => LA
                )
            [hobbies] => Array
                (
                    [0] => Crocheting
                    [1] => Hunting
                    [2] => Swimming
                )
        )
)
```

Reading `.xml` files in PHP

PHP provides a `simplexml_load_string` function that you can combine with the `file_get_contents` to read `.xml` files in your applications. For example, you can parse and use the content of a `.xml` file as an array in your code like this:

```php
if (file_exists('data.xml')) {
    $xmlString = file_get_contents('data.xml');
    $xml = simplexml_load_string($xmlString);
    $json = json_encode($xml);
    $array = json_decode($json, true);
    echo '<pre>';
    print_r($array);
    echo '</pre>';
} else {
    echo 'File does not exist';
}
```

The code above prints out the content of the `data.xml` file as an associative array like this:

```
Array
(
    [person] => Array
        (
            [0] => Array
                (
                    [name] => John Doe
                    [age] => 30
                    [email] => john@example.com
                )

            [1] => Array
                (
                    [name] => Jane Smith
                    [age] => 25
                    [email] => jane@example.com
                )

            [2] => Array
                (
                    [name] => Bob Johnson
                    [age] => 35
                    [email] => bob@example.com
                )

        )

)
```

Now that you understand how to read different types of files in PHP, let's explore how to write to files in the next section.

Writing and manipulating files in PHP

Many use cases require you to create and write files in your applications. The following sections will explore how to create, write, and manipulate files in PHP.

Creating a new file in PHP

Creating new files in PHP is relatively easy using the *fopen* function. For example, you can create a new *log.txt* file like this:

```
if (!file_exists('log.txt')) {
    $file = fopen("log.txt", "w");
    fclose($file);
} else {
    echo 'File exists already';
}
```

The code above checks whether the log file already exists before using the *fopen* function to create a new *log.txt* file.

Note: You can create any file using the above method; you only need to change the extension. For example, to make a *.json* file, you only need to add *.json* to the end of the file name.

Writing to an existing file in PHP

PHP allows you to write to files easily by opening a file in *w* mode. For example, you can write to the *log.txt* file you created in the previous section like this:

```
if (file_exists('log.txt')) {
    $file = fopen("log.txt", "w");
    fwrite(
        $file,
        "----- Log File -----

[2023-10-10 13:45:00] - Application started.
[2023-10-10 14:15:25] - User 'john_doe' logged in.
[2023-10-10 15:30:12] - Error: Unable to connect to the database.
"
    );
    fclose($file);
} else {
    echo 'File does not exist.';
}
```

The code above checks if the log file exists before using the *fopen* function with the *w* mode to write to the *log.txt* file. The *log.txt* file should now contain the following:

```
----- Log File -----

[2023-10-10 13:45:00] - Application started.
[2023-10-10 14:15:25] - User 'john_doe' logged in.
[2023-10-10 15:30:12] - Error: Unable to connect to the database.
```

Note: The code above would replace any content the file has in it with the given text in the code.

Appending to files in PHP

Appending to files in PHP means adding to the content in a file instead of overriding it. For example, to add to the log file, you can do the following:

```php
<?php

if (file_exists('log.txt')) {
    $file = fopen("log.txt", "a");
    fwrite(
        $file,
        "[2023-10-10 16:00:45] - File 'document.txt' successfully uploaded.
[2023-10-10 18:20:03] - User 'jane_smith' logged out.

----- End of Log -----"
    );
    fclose($file);
} else {
    echo 'File does not exist.';
}
```

The code above does the same thing as the previous one but with the *a* mode to add the content instead of overriding the existing content.

Writing to `.csv` files

PHP makes writing to a *`.csv`* file easy by providing you with the *`fputcsv`* function. For example, you can create and write to a *`data.csv`* file like so:

```php
$data = [['name', 'age', 'email'], ['Adams Adebayo', 22, 'adams@example.com'],
['Isreal Adesanya', 33, 'isreal@stylebender.com']];
$file = fopen("data.csv", "a");
foreach ($data as $row) {
    fputcsv($file, $row);
}
fclose($file);
```

The code above defines a *`data`* variable containing the data that should be written into the *`data.csv`* file, creates the file using the *`fopen`* function with the *`a`* mode, and uses a *`foreach`* loop and the *`fputcsv`* function to write the *`data`* variable to the created file.

Once the code is run, the *`data.csv`* file should now be in the folder with the following content:

```
name,age,email
"Adams Adebayo",22,adams@example.com
"Isreal Adesanya",33,isreal@stylebender.com
```

Writing to `.xml` files

PHP makes writing to a *.xml* file easy by providing the *file_put_contents* function. For example, you can create and write to a *data.xml* file like so:

```
$file = fopen("data.xml", "a");
$xml = '<?xml version="1.0" encoding="UTF-8"?>
        <data>
            <name>John</name>
            <age>30</age>
        </data>';
fwrite($file, $xml);
fclose($file);
```

The code above defines an *xml* variable containing the data that should be written into the *data.xml* file, creates the file using the *fopen* function with the *a* mode, and uses the *fwrite* function to write the *xml* variable into the created file.

Once the code is run, the *data.xml* file should now be in the folder with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
        <data>
            <name>John</name>
            <age>30</age>
        </data>
```

Deleting files in PHP

PHP provides an *unlink* function that you can use to delete files in your application. For example, you can delete the *data.xml* file you created in the previous section like this:

```
if (file_exists('data.xml'))
    $del = unlink('data.xml');
```

Once the code is run, the specified file will be deleted from the directory.

Note: Always check for the file's existence before deleting because the code above will return a warning if the specified file does not exist.

Let's explore how to work with directories in the next section.

Working with directories in PHP

PHP provides various functions that enable you to work with folders, commonly known as directories, in your applications. We will explore some of those in the following sections.

Creating a directory in PHP

Creating directories in PHP is quite straightforward using the *mkdir* function. For example, you can create a *sample* folder with three files inside it like this:

```
$dirPath = 'sample';
if (!file_exists($dirPath)) {
    mkdir($dirPath);
    fopen($dirPath . '/index.txt', 'w');
    fopen($dirPath . '/index.csv', 'w');
    fopen($dirPath . '/index.json', 'w');
}
```

The code above checks if the *sample* directory exists and creates it alongside three other files if it doesn't. Once the code is run, there should be a *sample* folder and three *index* files with different extensions inside it in your project directory.

List contents of a directory in PHP

Listing the contents of a given directory is a prevalent task in PHP. You can list the contents of the *sample* directory like this:

```
$dirPath = 'sample';
$contents = scandir($dirPath);
foreach ($contents as $item) {
    echo $item . '<br>';
}
```

The code above uses the *scandir* function with the *foreach* loop to list its content like so:

```
.
..
index.csv
index.json
index.txt
```

Note: The single dot represents the root folder, and the double dots represent the *sample* folder. Also, if subdirectories are in the given folder, they will also be listed.

You can filter to list only files like this:

```
$dirPath = 'sample';
$contents = scandir($dirPath);
foreach ($contents as $item) {
    if (is_file($dirPath . '/' . $item)) {
        echo $item . '<br>';
    }
}
```

The code above adds another check to the previous one to check if the item is a file using the *is_file* function before listing them.

You can also list only subfolders in the given directory by using the *is_dir* like this:

```
$dirPath = 'sample';
$contents = scandir($dirPath);
foreach ($contents as $item) {
    if (is_dir($dirPath . '/' . $item)) {
        echo $item . '<br>';
    }
}
```

Renaming directories in PHP

PHP allows you to rename directories in your applications with the *rename* function. For example, you can rename the *sample* directory like this:

```
$oldDirName = 'sample';
$newDirName = 'dataSamples';
if (is_dir($oldDirName)) {
    rename($oldDirName, $newDirName);
}
```

The code above checks if the *oldDirName* is a directory and renames it as *newDirName*.

Deleting directories in PHP

PHP provides a *rmdir* function that can be used to delete directories in your application. For example, you can delete a directory in your application like this:

```
if (is_dir("dataSamples")) {
    rmdir($dirPath);
}
```

The code above checks if the given string is a directory name and deletes it from the folder. However, the code above will return a warning telling you that the directory is not empty like so:

```
Warning: rmdir(dataSamples): Directory not empty in
/Users/Adams/Documents/phps/phpfileops/index.php on line 50
```

In the next section, let's explore how to delete a directory with all its content.

Deleting a directory with all its content in PHP

You can delete the *dataSamples* directory with all its content by creating a function like so:

```
function deleteFolder($dirPath)
{
    if (!is_dir($dirPath)) {
        return false;
    }
    // Open the directory for reading
    $dirHandle = opendir($dirPath);

    // Loop through the directory and delete its contents
    while (($file = readdir($dirHandle)) !== false) {
        if ($file !== '.' && $file !== '..') {
            $filePath = $dirPath . '/' . $file;

            // If it's a directory, recursively delete it
            if (is_dir($filePath)) {
                deleteFolder($filePath);
            } else {
                // If it's a file, delete it
                unlink($filePath);
            }
        }
    }

    // Close the directory handle
    closedir($dirHandle);

    // Delete the empty directory
    rmdir($dirPath);

    return true;
}
```

The code above creates a function that combines the functions we explored in the previous section and a `while` loop to open, read, and delete subdirectories and files inside the directory before deleting it. You can use the function above to delete the `dataSamples` folder and its content like so:

```
$folderToDelete = 'dataSamples';
if (deleteFolder($folderToDelete)) {
    echo 'Folder and its contents deleted successfully.';
} else {
    echo 'Failed to delete folder.';
}
```

The code above deletes the `dataSamples` directory and returns the following:

```
Folder and its contents deleted successfully.
```

Now that you understand how to read, write, and manipulate files and directories in PHP, let's explore the advanced file-handling tasks you would encounter when building PHP applications.

Advanced file operations in PHP

In this section, we will explore some of the different advanced file operations you might encounter when building PHP applications.

Create a ZIP archive of a directory

Creating a ZIP archive of folders is a common task you will come across when building PHP applications. You can create a ZIP archive of a folder like this:

```php
$sourceDir = 'sample';
$zipFileName = 'sample.zip';
$zip = new ZipArchive();
if ($zip->open($zipFileName, ZipArchive::CREATE) === true) {
    $files = new RecursiveIteratorIterator(new
RecursiveDirectoryIterator($sourceDir));
    foreach ($files as $file) {
        if (!$file->isDir()) {
            $filePath = $file->getRealPath();
            $relativePath = substr($filePath, strlen($sourceDir) + 1);
            $zip->addFile($filePath, $relativePath);
        }
    }
    $zip->close();
    echo 'Directory zipped successfully.';
} else {
    echo 'Failed to create ZIP archive.';
}
```

The code above uses the `ZipArchive`, `RecursiveIteratorIterator`, and `RecursiveDirectoryIterator` classes to create a ZIP archive of the given folder. Once the code is run, it will create a `sample.zip` file in the root folder and print out the following message:

```
Directory zipped successfully.
```

Here's a `log.txt` that I'll be using to demonstrate the following sections:

```
[2023-10-10 13:45:00] - Application started.
[2023-10-10 14:15:25] - User 'john_doe' logged in.
[2023-10-10 15:30:12] - Error: Unable to connect to the database.
[2023-10-10 16:00:45] - File 'document.txt' successfully uploaded.
[2023-10-10 18:20:03] - User 'jane_smith' logged out.
[2023-10-10 08:00:00] - Application started.
[2023-10-10 08:15:25] - User 'john_doe' logged in.
[2023-10-10 08:30:12] - Error: Unable to connect to the database.
[2023-10-10 09:00:45] - File 'document.txt' successfully uploaded.
[2023-10-10 09:30:03] - User 'jane_smith' logged out.
[2023-10-10 10:05:15] - Information: 5 new emails received.
[2023-10-10 11:20:45] - Warning: High CPU usage detected.
[2023-10-10 11:45:10] - User 'admin' made changes to the system settings.
[2023-10-10 12:10:33] - Database backup completed successfully.
[2023-10-10 13:25:50] - User 'alice' updated her profile.
[2023-10-10 14:00:02] - Information: 10 new user registrations.
[2023-10-10 14:30:19] - File 'report.pdf' downloaded by user 'bob'.
```

```
[2023-10-10 15:15:37] - Warning: Low disk space on server.
[2023-10-10 16:05:55] - User 'charlie' changed their password.
[2023-10-10 16:35:11] - Error: Server crashed, restart required.
[2023-10-10 17:10:28] - Information: Daily backup completed.
[2023-10-10 17:45:46] - User 'david' created a new project.
[2023-10-10 18:20:00] - File 'presentation.ppt' uploaded by user 'emily'.
[2023-10-10 18:45:21] - User 'frank' logged in.
[2023-10-10 19:15:40] - Warning: Network connection lost.
[2023-10-10 19:45:59] - User 'grace' sent a message to 'harry'.
[2023-10-10 20:10:18] - Information: Server maintenance in progress.
[2023-10-10 20:30:32] - User 'ian' logged out.
[2023-10-10 21:05:47] - Error: Unable to load web page.
[2023-10-10 21:25:59] - File 'data.csv' exported by user 'jack'.
[2023-10-10 22:00:10] - Information: 3 new comments on the blog.
[2023-10-10 22:30:22] - User 'kate' updated their profile picture.
[2023-10-10 23:00:34] - Warning: Unusual login activity detected.
[2023-10-10 23:20:45] - User 'lucy' logged in from a new device.
[2023-10-10 23:45:57] - Information: Scheduled tasks completed.
```

Read and display the first `N` lines of a file

You can easily get the first N lines of a file in PHP using the following code:

```
$file = fopen('log.txt', 'r');
$linesToRead = 10;
$lineCount = 0;
while (($line = fgets($file)) !== false && $lineCount < $linesToRead) {
    echo $line . '<br>';
    $lineCount++;
}
fclose($file);
```

The code above returns the first ten lines of the `log.txt` file.

Read and display the last N lines of a file

You can easily get the last N lines of a file in PHP using the following code:

```
function displayLastNLines($filename, $n)
{
    $lines = [];
    $file = new SplFileObject($filename, 'r');

    $file->seek(PHP_INT_MAX);
    $totalLines = $file->key() + 1;

    if ($totalLines < $n) {
        $file->seek(0);
    } else {
        $file->seek($totalLines - $n);
    }

    while (!$file->eof()) {
        $lines[] = $file->current();
        $file->next();
    }

    foreach (array_slice($lines, -$n) as $line) {
        echo $line;
        echo '<br>';
    }
}

$filename = 'log.txt';
$numberOfLines = 10;
displayLastNLines($filename, $numberOfLines);
```

The code above creates a function that takes in a file name and a number of lines and returns data based on the given number from the file.

Read and search for a specific string in a file

You can search for words in a given file in PHP like this:

```php
function findAllInstancesInFile($filename, $searchString)
{
    $fileContents = file_get_contents($filename);
    $lines = explode(PHP_EOL, $fileContents);
    $foundInstances = [];

    foreach ($lines as $line) {
        if (strpos($line, $searchString) !== false) {
            $foundInstances[] = $line;
        }
    }

    return $foundInstances;
}

$filename = 'log.txt';
$searchString = 'database';
$foundInstances = findAllInstancesInFile($filename, $searchString);

if (!empty($foundInstances)) {
    echo "Found instances of '$searchString' in the file: <br>";
    foreach ($foundInstances as $instance) {
        echo $instance;
        echo '<br>';
    }
} else {
    echo "No instances of '$searchString' found in the file.";
}
```

The code above creates a function that searches for a given string in the given file and returns an array of the found instances.

Read and display the file size

You can retrieve the size of a file in PHP using the *`filesize`* function like this:

```php
$fileSize = filesize('log.txt');
echo "File size: $fileSize bytes";
```

The code above returns the size of the *`log.txt`* file like so:

```
File size: 2127 bytes
```

Read and display file metadata

Retrieving metadata of files is also straightforward in PHP with the `stat` function:

```
$filePath = 'log.txt';
$fileInfo = stat($filePath);
echo "File Size: {$fileInfo['size']} bytes" . '<br>';
echo "Last Modified: " . date('Y-m-d H:i:s', $fileInfo['mtime']) . '<br>';
echo "Last Accessed: " . date('Y-m-d H:i:s', $fileInfo['atime']) . '<br>';
echo "Owner: {$fileInfo['uid']}" . '<br>';
echo "Group: {$fileInfo['gid']}" . '<br>';
echo "File Permissions: {$fileInfo['mode']}" . '<br>';
echo "Inode Number: {$fileInfo['ino']}" . '<br>';
echo "Number of Links: {$fileInfo['nlink']}" . '<br>';
```

The code above returns the following information about the file:

```
File Size: 2128 bytes
Last Modified: 2023-10-10 21:22:47
Last Accessed: 2023-10-10 21:22:48
Owner: 501
Group: 20
File Permissions: 33188
Inode Number: 15392607
Number of Links: 1
```

Note: The result will be different depending on the given file.