

Ch6: FILE HANDLING

File path & permissions:

File permissions specify what a user can do with a file, e.g., reading, writing, or executing it. Notice that PHP automatically grants appropriate permissions behind the scenes.

For example, if you create a new file for writing, PHP automatically grants the read and write permissions.

PHP provides some useful functions for checking and changing the file permissions.

Checking file permissions

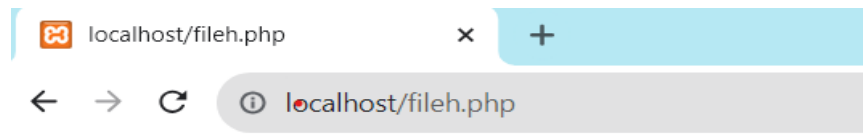
PHP has three handy functions that check file permissions:

- **is_readable()** function returns true if the file exists and is readable; otherwise, it returns false.
- **is_writable()** function returns true if the file exists and is writable; otherwise, it returns false.
- **is_executable()** function returns true if the file exists and executable; otherwise, it returns false.

Let's take a look at the following example:

```
<?php
$filename = 'pst list.docx';
$functions = [
    'is_readable',
    'is_writable',
    'is_executable'
];
foreach ($functions as $f)
{
    echo $f($filename) ? 'The file ' . $filename . $f : '<br>';
    echo "<br>";
}
```

Output:



The file pst list.docxis_readable
The file pst list.docxis_writable

Besides those functions, PHP also provides the **fileperms()** function that returns an integer, which represents the permissions set on a particular file. For example:

```
<?php

$permissions = fileperms('readme.txt');

echo substr(sprintf('%o', $permissions), -4);
```

Changing file permissions

To change the file permission or mode, you use the `chmod()` function:

```
chmod ( string $filename , int $permissions ) : bool
```

The `chmod()` function has two parameters:

\$filename is the file that you want to change the permissions.

\$permissions parameter consists of three octal number components that specify access restrictions for the owner, the user group in which the owner is in, and everyone else in this sequence.

The `chmod()` function returns true on success or false on failure.

The permissions argument is represented by an octal number that contains three digits:

1. The first digit specifies what the owner of the file can read, write, or execute the file.
2. The second digit specifies what the user group in which the owner is in can read, write, or execute the file.

3. The third digit specifies what everyone else can read, write, or execute the file.

The following table illustrates the value of each digit that represents the access permission for particular users (owner, user group, or everyone else) :

Value	Permission
0	cannot read, write or execute
1	can only execute
2	can only write
3	can write and execute
4	can only read
5	can read and execute
6	can read and write
7	can read, write and execute

The following example sets permission that the only owner can read and write a file, everyone else only can read the file:

```
<?php
$filename = 'pst list.docx';
$functions = [
    'is_readable',
    'is_writable',
    'is_executable'
];
foreach ($functions as $f)
{
    echo $f($filename) ? 'The file ' . $filename . $f : '<br>';
    echo "<br>";
}
?>
<?php
$filename = 'pst list.docx';
```

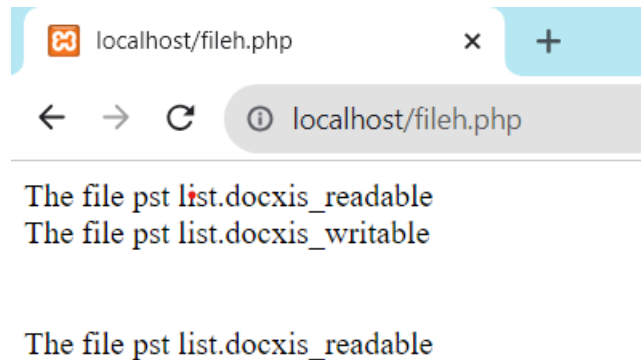
```

chmod($filename, 0444);

foreach ($functions as $f)
{
    echo $f($filename) ? 'The file ' . $filename . $f : '<br>';
    echo "<br>";
}
?>

```

Output:



Notice that we put 0 before 644 to instruct PHP to treat it as an octal number.

Summary

Use the `is_readable()`, `is_writable()`, `is_executable()` to check if a file exists and readable, writable, and executable.

Use the `chmod()` function to set permissions for a file.

Displaying directory contents

PHP Directory Functions

Function	Description
<u>chdir()</u>	Changes the current directory
<u>chroot()</u>	Changes the root directory

<u>closedir()</u>	Closes a directory handle
<u>dir()</u>	Returns an instance of the Directory class
<u>getcwd()</u>	Returns the current working directory
<u>opendir()</u>	Opens a directory handle
<u>readdir()</u>	Returns an entry from a directory handle
<u>rewinddir()</u>	Resets a directory handle
<u>scandir()</u>	Returns an array of files and directories of a specified directory

scandir():

The **scandir()** function in PHP is an inbuilt function that is used to return an array of files and directories of the specified directory.

The scandir() function lists the files and directories which are present inside a specified path. The directory, stream behavior, and sorting_order of the files and directories are passed as a parameter to the scandir() function and it returns an array of filenames on success, or false on failure.

Syntax:

```
scandir(directory, sorting_order, context);
```

Parameters: The scandir() function in PHP accepts 3 parameters that are listed below:

- **directory:** It is a mandatory parameter that specifies the path.
- **sorting_order:** It is an optional parameter that specifies the sorting order. Alphabetically ascending order (0) is the default sort order. It can be set to SCANDIR_SORT_DESCENDING or 1 to sort in alphabetically descending order, or SCANDIR_SORT_NONE to return the result unsorted.
- **context:** It is an optional parameter that specifies the behavior of the stream.

Return Value: It returns an array of filenames on success, or false on failure.

Errors

- The scandir() function throws an error of level E_WARNING if the directory specified is not a directory.

Example 1: The below example illustrate the **scandir()** function that will scan the files & return value will be in the ascending order.

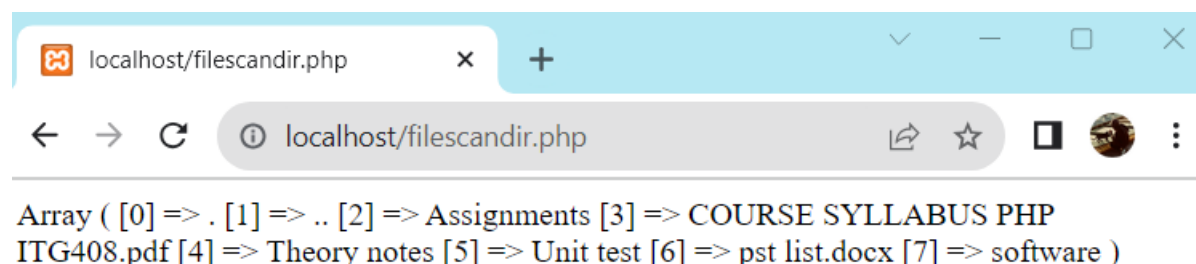
```
<?php

// Specifying directory
$mydir = 'C:\Users\OneDrive\Desktop\KHAN\WET TY';

// Scanning files in a given directory in ascending order
$myfiles = scandir($mydir);

// Displaying the files in the directory
print_r($myfiles);
?>
```

Output:



```
Array ( [0] => . [1] => .. [2] => Assignments [3] => COURSE SYLLABUS PHP
ITG408.pdf [4] => Theory notes [5] => Unit test [6] => pst list.docx [7] => software )
```

Example 2: This example illustrates the **scandir()** function that will scan the files & return value will be in the descending order.

```
<?php

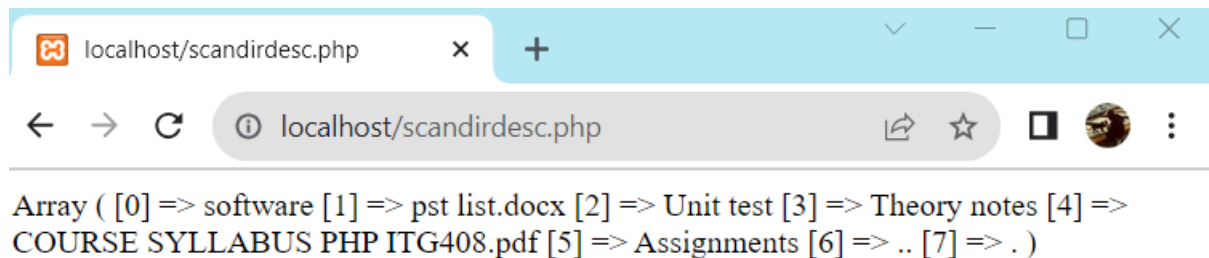
// Specifying directory
```

```
$mydir = 'C:\Users\OneDrive\Desktop\KHAN\WET TY';

// Scanning files in a given directory in ascending order
$myfiles = scandir($mydir,1);

// Displaying the files in the directory
print_r($myfiles);
?>
```

Output:



```
Array ( [0] => software [1] => pst list.docx [2] => Unit test [3] => Theory notes [4] =>
COURSE SYLLABUS PHP ITG408.pdf [5] => Assignments [6] => .. [7] => . )
```

Example 3: This example illustrates the **scandir()** function that will scan the files & return value will be in the unsorted order.

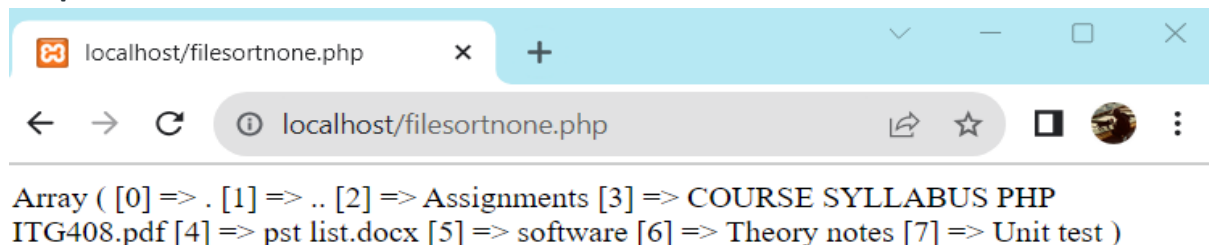
```
<?php

// Specifying directory
$mydir = 'C:\Users\OneDrive\Desktop\KHAN\WET TY';

// Scanning files in a given directory in ascending order
$myfiles = scandir($mydir,SCANDIR_SORT_NONE);

// Displaying the files in the directory
print_r($myfiles);
?>
```

Output:



```
Array ( [0] => . [1] => .. [2] => Assignments [3] => COURSE SYLLABUS PHP
ITG408.pdf [4] => pst list.docx [5] => software [6] => Theory notes [7] => Unit test )
```

Working with fopen() and fclose()

This chapter will explain following functions related to files –

- Opening a file

- Reading a file
- Writing a file
- Closing a file

Opening and Closing Files

The PHP **fopen()** function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Files modes can be specified as one of the six options in this table.

Sr.No	Mode & Purpose
	r
1	Opens the file for reading only. Places the file pointer at the beginning of the file.
	r+
2	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
	w
3	Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
	w+
4	Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
	a
5	Opens the file for writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.
	a+
6	Opens the file for reading and writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.

If an attempt to open a file fails then **fopen** returns a value of **false** otherwise it returns a **file pointer** which is used for further reading or writing to that file.

After making a changes to the opened file it is important to close it with the **fclose()** function. The **fclose()** function requires a file pointer as its argument and then returns **true** when the closure succeeds or **false** if it fails.

Reading a file

Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The files length can be found using the **filesize()** function which takes the file name as its argument and returns the size of the file expressed in bytes.

So here are the steps required to read a file with PHP.

- Open a file using **fopen()** function.
- Get the file's length using **filesize()** function.
- Read the file's content using **fread()** function.
- Close the file with **fclose()** function.

The following example assigns the content of a text file to a variable then displays those contents on the web page.

```
<html>
<head>
    <title>Reading a file using PHP</title>
</head>
<body>
    <?php
        $filename = "filetest.docx";
        $file = fopen( $filename, "r" );

        if( $file == false ) {
            echo ( "Error in opening file" );
            exit();
        }

        $filesize = filesize( $filename );
        $filetext = fread( $file, $filesize );
```

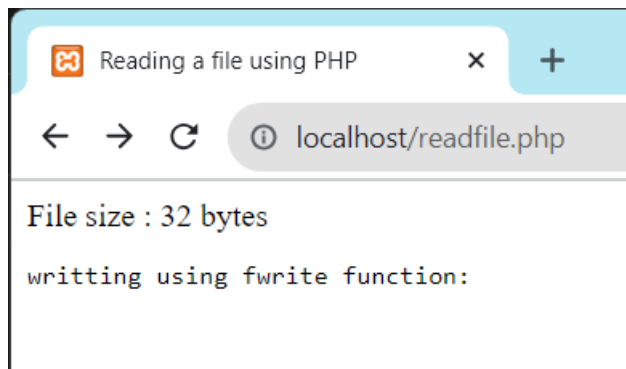
```

        fclose( $file );

        echo ( "File size : $filesize bytes" );
        echo ( "<pre>$filetext</pre>" );
    ?>
</body>
</html>

```

It will produce the following result –



Writing a file

A new file can be written or text can be appended to an existing file using the PHP **fwrite()** function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included, writing would stop after the specified length has been reached.

The following example creates a new text file then writes a short text heading inside it. After closing this file its existence is confirmed using **file_exists()** function which takes file name as an argument

```

<?php
$filename = "filetest.docx";
$file = fopen( $filename, "w" );
if( $file == false ) {
    echo ( "Error in opening new file" );
    exit();
}
fwrite( $file, "writting using fwrite function:\n" );

```

```

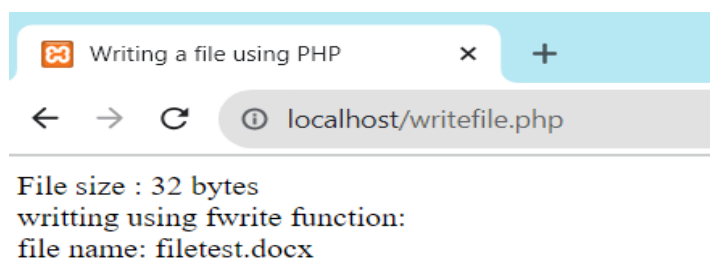
    fclose( $file );
?>

<html>
  <head>
    <title>Writing a file using PHP</title>
  </head>
  <body>
    <?php
      $filename = "filetest.docx";
      $file = fopen( $filename, "r" );
      if( $file == false ) {
        echo ( "Error in opening file" );
        exit();
      }
      $filesize = filesize( $filename );
      $filetext = fread( $file, $filesize );
      fclose( $file );

      echo ( "File size : $filesize bytes" );
      echo ( "<br>" );
      echo ( "$filetext" );
      echo ( "<br>" );
      echo("file name: $filename");
    ?>
  </body>
</html>

```

It will produce the following result –



PHP Append Text

You can append data to a file by using the "a" mode. The "a" mode appends text to the end of the file, while the "w" mode overrides (and erases) the old content of the file.

In the example below we open our existing file " `filetest.docx` ", and append some text to it:

Example

```
<html>
  <head>
    <title>Appending a file using PHP</title>
  </head>
  <body>
    <?php
      $filename = "filetest.docx";
      $file = fopen($filename, "a");
      if( $file == false )
      {
        echo ( "Error in opening file" );
        exit();
      }

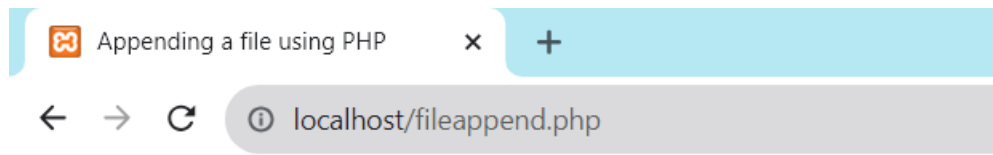
      $txt = "TYIT";
      fwrite($file, $txt);

      $txt = "IF1 \n";
      fwrite($file, $txt);
      fclose( $file );

      $file = fopen($filename, "r");
      if( $file == false )
      {
        echo ( "Error in opening file" );
        exit();
      }
      $filesize = filesize( $filename );
      $filetext = fread( $file, $filesize );
      fclose( $file );

      echo ( "File size : $filesize bytes" );
      echo ( "<pre>$filetext</pre>" );
    ?>
  </body>
</html>
```

If we now open the " `filetest.docx` " file, we will see that `TYIT` and `IF1` is appended to the end of the file



File size : 181 bytes •

```
writting using fwrite function : from if1
TYITIF1TYITIF1TYITIF1TYITIF1TYITIF1
TYITIF1 /nTYITIF1 /nTYITIF1 /nTYITIF1 /nTYITIF1 /nTYITIF1
TYITIF1
TYITIF1
TYITIF1
TYITIF1 IF2 IF3
```

File system housekeeping

- Copying file
- Renaming file
- Deleting file

Copying file

The **copy()** function in PHP is used to copy a file from source to target or destination directory. It makes a copy of the source file to the destination file and if the destination file already exists, it gets overwritten. The copy() function returns true on success and false on failure.

Syntax:

bool copy(string \$source, string \$destination, resource \$context)

Parameters: This function uses three parameters source, destination and context which are listed below:

- **\$source:** It specifies the path of the source file.
- **\$destination:** It specifies the path of the destination file or folder.
- **\$context:** It specifies the context resource created with stream_context_create() function. It is optional parameter.

Return: It returns a boolean value, either true (on success) or false (on failure).

Example:

```
<?php
// file copy
// Store the path of source file
```

```

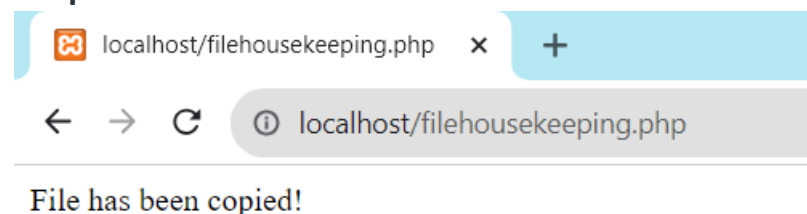
$source = 'C:\xampp\htdocs\test1.docx';

// Store the path of destination file
$destination = 'C:\xampp\htdocs\test2.docx';

// Copy the file from C:\xampp\htdocs\test1.docx
// to C:\xampp\htdocs\test2.docx
// directory
if( !copy($source, $destination) ) {
    echo "File can't be copied! \n";
}
else {
    echo "File has been copied! \n";
}
?>

```

Output:



Renaming file

The `rename()` function in PHP is an inbuilt function which is used to rename a file or directory. It makes an attempt to change an old name of a file or directory with a new name specified by the user and it may move between directories if necessary.

If the new name specified by the user already exists, the `rename()` function overwrites it. The old name of the file and the new name specified by the user are sent as parameters to the `rename()` function and it returns True on success and a False on failure.

Syntax:

`rename(oldname, newname, context)`

Parameters Used:

The `rename()` function in PHP accepts three parameter.

1. **oldname** : It is a mandatory parameter which specifies the old name of the file or directory.

2. **newname** : It is a mandatory parameter which specifies the new name of the file or directory.
3. **context** : It is an optional parameter which specifies the behavior of the stream.

Return Value:

It returns True on success and a False on failure.

Program 1

```
<?php
//file renaming

// Old Name Of The file
$old_name = "pst list.docx" ;

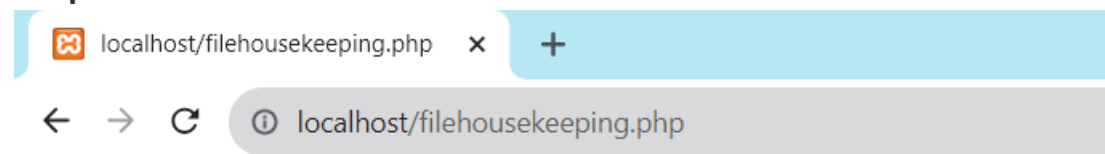
// New Name For The File
$new_name = "test1renamed.docx" ;

// using rename() function to rename the file
rename( $old_name, $new_name) ;

//display a msg
echo "\n Successfully Renamed $old_name to $new_name \n" ;

?>
```

Output:



File has been copied! Successfully Renamed pst list.docx to test1renamed.docx

Program 2

```
<?php
//renaming and checking for existing file
// Old Name Of The file
$old_name = "filetest.docx" ;

// New Name For The File
```

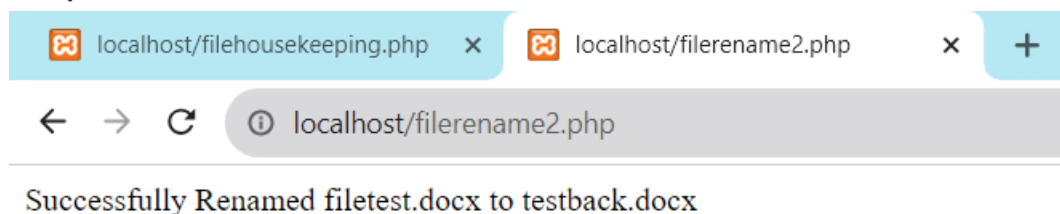
```

$new_name = "testback.docx" ;

// Checking If File Already Exists
if(file_exists($new_name))
{
    echo "Error While Renaming $old_name" ;
}
else
{
    if(rename( $old_name, $new_name))
    {
        echo "Successfully Renamed $old_name to $new_name" ;
    }
    else
    {
        echo "A File With The Same Name Already Exists" ;
    }
}
}

```

Output:



Deleting file:

To delete a file by using PHP is very easy. Deleting a file means completely erase a file from a directory so that the file is no longer exist. PHP has an unlink() function that allows to delete a file. The PHP unlink() function takes twoparameters *\$filename* and *\$context*.

Syntax:

```
unlink( $filename, $context );
```

Program 1: This program uses unlink() function to remove file from directory. Suppose there is a file named as “gfg.txt”

```

<?php
// PHP program to delete a file named pst list.docx
// using unlink() function

$file_pointer = "pst list.docx";

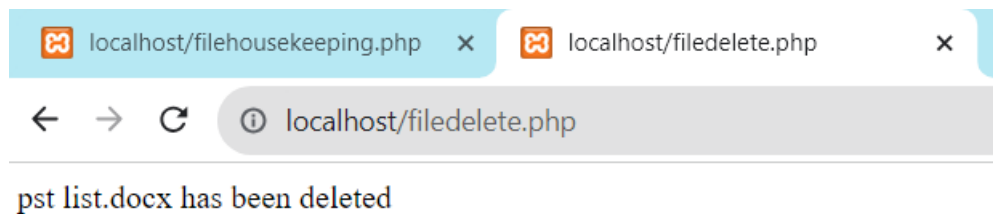
```



```
// Use unlink() function to delete a file
if (!unlink($file_pointer)) {
    echo ("{$file_pointer} cannot be deleted due to an error");
}
else {
    echo ("{$file_pointer} has been deleted");
}

?>
```

Output:



Note: If the file does not exist then it will display an error.