TEXT FILES IN PHP

PHP OFFERS A BUNCH OF FUNCTIONS TO HANDLE FILES.

IN THIS LESSON WE WILL SEE HOW TO WORK WITH TEXT FILES, AND HOW TO DO OPERATIONS LIKE:

- OPEN A FILE
- CREATE A FILE
- READ
- WRITE

TEXT FILES (EVEN BETTER JSON AND XML FILES) CAN BE AN ALTERNATIVE TO SQL DATABASES WHEN THE AMOUNT OF DATA IS SMALL.

OPENING A FILE

WE CAN TRY TO OPEN A FILE WITH THE FUNCTION `fopen()`

```
$f=fopen(filename, mode);   // $f is the file handler
```

MODE CAN BE:

`r`	Read-only. Pointer placed at the beginning of the file. Gives a warning if the file doesn't exist.
`w`	Write-only. Pointer placed at the beginning of the file. The file is created every time (it will be emptied if exists).
`a`	Write-only. Pointer placed at the end of the file. Creates the file if it doesn't exist.
`x`	Creates a file only for writing. Pointer placed at the beginning of the file. Gives a warning if the file already exists.
`c`	Write-only. Pointer placed at the beginning of the file. Creates the file if it doesn't exist (it will not be emptied if exists).

OPENING A FILE

OTHER MODES:

`r+`	Like 'r' but the file is opened for reading and writing
`w+`	Like 'w' but the file is opened for reading and writing
`a+`	Like 'a' but the file is opened for reading and writing.
`x+`	Like 'x' but the file is created for reading and writing.
`c+`	Like 'c' but the file is opened for reading and writing.

```
fclose($f);
```

— Closes an open file pointer

OPENING A FILE

WE CAN CHECK IF THE FILE EXISTS WITH THE FUNCTION `file_exists()`

```
if(!file_exists("samplefile.txt")) {
      $f = fopen("samplefile.txt","w+");
}
else {
      $f = fopen("samplefile.txt","a+");
}
```

IF THE FILE `samplefile.txt` DOESN'T EXIST, IT WILL BE CREATED.

IF THE FILE EXISTS, IT WILL BE OPENED FOR READING AND WRITING.

IF A PATH IS NOT PROVIDED, THE FILE WILL BE CREATED (OR SOUGHT) IN THE SAME FOLDER WHERE THE SCRIPT IS.

READING A TEXT FILE

READING A FILE IN A SINGLE OPERATION:

- `readfile()` opens a file and directly outputs its contents.
- `file_get_contents()` reads the file into a single string
- `file()` reads each line into an array.

IF WE HAVE THE FILE `samplefile.txt` WITH THESE CONTENTS:

```
Line 1
Line 2
Line 3
Line 4
Line 5
```

READING A TEXT FILE

READING A FILE IN A SINGLE OPERATION:

```
<?php
    readfile("samplefile.txt");
?>
```

OUTPUT:

Line 1 Line 2 Line 3 Line 4 Line 5

READING A TEXT FILE

READING A FILE IN A SINGLE OPERATION:

```php
<?php
    echo file_get_contents("samplefile.txt");
?>
```

Line 1 Line 2 Line 3 Line 4 Line 5

```php
<?php
   echo nl2br(file_get_contents("samplefile.txt"));
?>
```

Line 1
Line 2
Line 3
Line 4
Line 5

The function `nl2br` converts newline characters into `<br/>` HTML tags

READING A TEXT FILE

READING A FILE IN A SINGLE OPERATION:

```
<?php
    $array = file("samplefile.txt");
    print_r($array);
?>
```

EMPTY LINES WITH A NEWLINE CHARACTER

```
Array ( [0] => Line 1 [1] => Line 2 [2] => Line 3 [3] => Line 4 [4] => Line 5 [5] => [6] => [7] => )
```

```
<?php
  $array = file("samplefile.txt",
   FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
  print_r($array);
?>
```

REMOVES NEWLINES CHARACTERS AND IGNORES EMPTY LINES

REMOVES ALL LINES NOT JUST THE END

```
Array ( [0] => Line 1 [1] => Line 2 [2] => Line 3 [3] => Line 4 [4] => Line 5 )
```

READING A TEXT FILE

WE HAVE SEEN SOME FUNCTIONS THAT READ THE WHOLE FILE INTO MEMORY IN A SINGLE OPERATION.

WITH VERY LARGE FILES IT IS BETTER TO USE FUNCTIONS THAT PROCESS ONLY A PART OF THE FILE AT A TIME (A LINE, A WORD, A CHARACTER...)

BESIDES THE `fopen()` FUNCTION TO OPEN THE FILE, WE HAVE THESE OTHER FUNCTIONS TO READ A FILE:

READING A TEXT FILE

READING A CERTAIN NUMBER OF BYTES

```
fread(file_handler, number_of_bytes)
```

```
$f=fopen(filename, r);
$content=fread($f, filesize(filename)); // reads the whole file
```

READING ONLY A LINE

```
fgets(file_handler)
```

```
$f=fopen(filename, r);
$line=fgets($f);   // first fgets reads the first line,
                   // next fgets reads next line, and so on
                   //  With the function feof() we detect the end of file

while(!feof($f)) {$line=fgets($f); … }
```

READING A CHAR AT THE TIME `fgetc(fitxer)`

READING A CSV FILE

The CSV format

- Allows data from a table (tabulated data) to be stored in a plain text file.
- Uses a delimiter to separate columns and a newline to separate rows.
- The default CSV delimiter is the comma – but other characters may be used.
- Is supported by all major spreadsheet and database applications.
- CSV files are typically given a .CSV or .TXT extension.

LOOK AT THE FILE agenda.txt in next page.

READING A CSV FILE

```
fgetcsv(file_handler, length, delimiter, enclosurechar)
```

- by default, the length is 0 (reads until end of line)
- use enclosurechar only if every column is surrounded by a enclosure char
  "Johnny", "Smith",…
- Reads rows just like fgets(), but returns an array of delimited fields instead of a single string.
- Starts reading at the file pointer and stops reading when it gets to a \n or EOF character.
- Moves the file pointer to the beginning of the next line after each read.

READING A CSV FILE

EXAMPLE OF READING A CSV FILE . WE HAVE THE FILE agenda.txt

"John","Smith","john@gmail.com","6650403234"
"Martha","Ford","martha@gmail.com","65345235"
"David","Garcia","dgarcia@gmail.com","69823422"

WE READ EACH LINE INTO AN ARRAY:

```
$file=fopen("agenda.txt","r");
while (!feof($file)) {
      $data=fgetcsv($file,0,',','"');
      print_r($data);
}
```

```
Array ( [0] => John [1] => Smith [2] => john@gmail.com [3] => 6650403234 )
Array ( [0] => Martha [1] => Ford [2] => martha@gmail.com [3] => 65345235 )
Array ( [0] => David [1] => Garcia [2] => dgarcia@gmail.com [3] => 69823422 )
```

WRITING INTO A TEXT FILE

WE CAN USE `fwrite(file_handler, content)` TO WRITE CONTENT INTO A TEXT FILE.

IF THE FILE HAS BEEN OPENED WITH "w" MODE, `fwrite` DELETES ANY EXISTING CONTENT BEFORE WRITING THE NEW ONE

IF THE FILE HAS BEEN OPENED WITH "a" MODE, `fwrite` APPENDS THE NEW CONTENT AT THE END OF THE FILE

IF WE WRITE A LINE AT THE TIME, IT MUST END WITH THE CONSTANT `PHP_EOL`

`fputs(file_handler, content)` IS A SYNONYM FOR `fwrite`

WRITING INTO A TEXT FILE

```
$file=fopen("append.txt","w");  // CREATES THE FILE
fwrite($file, "This is the first line".PHP_EOL);
fwrite($file, "And this is the second line".PHP_EOL);
fclose($file);
```

```
This is the first line
And this is the second line
```

```
$file=fopen("append.txt","a"); // CREATES THE FILE IF NOT EXISTS
fwrite($file, "This should be the third line".PHP_EOL);
fclose($file);
```

```
This is the first line
And this is the second line
This should be the third line
```

WRITING INTO A TEXT FILE

THE FUNCTION `fflush(file_handler)` FORCES A WRITE OF THE BUFFERED OUTPUT TO THE PHYSICAL FILE POINTED TO BY FILE_HANDLER

MOST LIKELY IT WON'T BE NECESSARY, BUT MAY BE USEFUL IF WE HAVE TROUBLES UPDATING THE CONTENT OF THE FILE

```php
<?php
$filename = 'testing.txt';

$file = fopen($filename, 'a+');
fwrite($file, 'a new line'.PHP_EOL);
fflush($file);
fclose($file);
?>
```

MOVING THE FILE POINTER

`rewind(fitxer)` MOVES THE POINTER TO THE BEGINNING OF THE FILE

`fseek()` MOVES THE POINTER A NUMBER OF BYTES, FORWARD (WITH A POSITIVE NUMBER) OR BACKWARDS (WITH A NEGATIVE NUMBER)

BY DEFAULT, THE MOVEMENT IS RELATIVE TO THE BEGINNING OF THE FILE. WE CAN USE `fseek` TO MAKE THE MOVEMENT RELATIVE TO THE END OF FILE, OR TO AN SPECIFIED POSITION

```
fseek($f, 0); // moves the pointer to the beginning of the file
fseek($f, 0,SEEK_END); // moves the pointer to the end of the file
fseek($f,25); // moves the pointer 25 bytes from the beginning of the file
fseek($f,25,SEEK_CUR); // moves the pointer 25 bytes from the current position
fseek($f,-25,SEEK_END);// moves the pointer 25 bytes backwards from the end of
                       // the file
```

In append mode (a or a+), content is always written to the end of the file regardless of the pointer's current position.

MOVING THE FILE POINTER

```
$file=fopen("append.txt","r+");
// the file is opened for reading and writing
// but the pointer is at the beginning of the file
fseek($file,0,SEEK_END);
// we move the pointer to the end of the file
fwrite($file, "This line should be placed at the end".PHP_EOL);
fclose($file);
```

```
This is the first line
And this is the second line
This should be the third line
This line should be place at the end
```

MOVING THE FILE POINTER

```
$file=fopen("append.txt","r+");
// the file is opened for reading and writing
// but the pointer is at the beginning of the file
// what happens if we forget to move the pointer?
fwrite($file, "This line should be placed at the end".PHP_EOL);
fclose($file);
```

This is the first line
And this is the second line
This should be the third line

→

```
This line should be placed at the end of the file

This should be the third line
```

WITHOUT MOVING THE POINTER TO THE END OF THE FILE, THE FIRST LINE HAS BEEN OVERWRITTEN, AND THE SECOND LINE HAS DISAPPEARED

LOCKING A FILE

TO LOCK A FILE BEFORE WORKING WITH IT, IN ORDER TO GET A SHARED OR EXCLUSIVE LOCK FOR READING OR WRITING, WE MUST OPEN THE FILE IN 'c' MODE

```
$f=fopen(filename,'c'); // OPENS THE FILE, OR CREATES IF NOT EXISTS
```

AND THEN WE GET THE LOCK BY USING THE `flock()` FUNCTION

- `LOCK_SH` acquires a shared lock for reading.
- `LOCK_EX` acquires an exclusive lock for writing.
- `LOCK_UN` releases the lock.

```
flock($file, LOCK_EX);   // LOCKS THE FILE FOR EXCLUSIVE WRITING
```

flock() places the internal pointer at the beginning of the file. We need to move the pointer to the end of the file (or delete the existing content) before writing (with fseek).

FUNCTION `stat()`

THE FUNCTION `stat(string $filename)` GETS STATISTICS ABOUT THE FILE LIKE DEVICE NUMBER, USERID AND GROUPID (IN LINUX), SIZE IN BYTES, TIME OF LAST ACCESS, TIME OF LAST MODIFICATION...

THE FUNCTION `fstat(file_handler)` DOES EXACTLY THE SAME BUT THE PARAMETER IS THE FILE HANDLER INSTEAD OF THE FILE NAME

FUNCTIONS `implode()` AND `explode()`

THE FUNCTION `explode(delimiter, string)` BREAKS A STRING INTO AN ARRAY AND THE DELIMITER CHARACTER SPECIFYING WHEN TO BREAK THE STRING

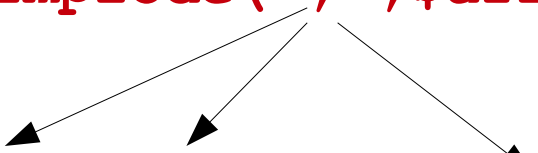
```
<?php
$str = "Hello you everyone";
print_r(explode(" ",$str));   // the delimiter is a blank character
?>
```

```
array ( [0] => Hello [1] => you [2] => everyone
```

FUNCTIONS `implode()` AND `explode()`

THE FUNCTION `implode(delimiter, array)` CONVERTS AN ARRAY INTO A STRING, WITH THE ARRAY ELEMENTS SEPARATED IN THE STRING BY THE DELIMITER CHARACTER

```
<?php
$arr = array('April','June','September','November');
echo implode(",",$arr);
?>
```



```
April,June,September,November
```