Working with Files and Processes in PHP

Files, Directories, Permissions

Mario Peshev

Technical Trainer

http://peshev.net

Software University

http://softuni.bg

Table of Contents

SOFTWARE UNIVERSITY FOUNDATION

Working with Files

Opening, Reading, and Writing

Opening a file

- Open a file and return the associated resource:

```
fopen($name, $mode, $use_include_path, $context)
```

- **$name** – the file to open
- **$mode** – the mode of file opening:
  - "w" – open for writing, clear all content
  - "r" – open for reading
  - "a" – open for writing, keep content, append
  - "x" – open new file for writing, if file exists – fail

Opening a file (2)

- All file modes can be used with a "+"
  - Open the file for both reading and writing
  - Examples: "r+", "w+", "a+", "x+"
- All file modes can be used with a "b" flag
  - Safe binary access
  - Used only on Windows-like systems which make difference between text files and binary files
  - Examples: "rb", "wb", "ab", "xb"

Opening a file (3)

- **$use_include_path** – optional
  - If true, the file will be searched in the PHP **include_path**
- **$context** – optional
  - Additional settings for writing streams
- Returns:
  - On success, a pointer to the file resource
  - On failure, **false**

```
// Example – open “text.txt” for reading, binary-safe
$file = fopen(“text.txt”, “rb”);
```

Reading from a file

- Read the specified number of characters from a file resource:

```
fread($resource, $length)
```

- **`$resource`** – a pointer to a file
  - The same type that `fopen()` returns
- **`$length`** – the number of bytes to read
  - 8192 (8 KB) by default
  - Reading stops when **`$length`** bytes have been read

Reading from a file (2)

- Returns:
  - On success, the read bytes (as string)
  - On failure, **`false`**
- Moves the internal file pointer by **`$length`** bytes

```
// Example – open “text.txt” for binary-safe reading
// and read the first 20 characters
$file = fopen(“text.txt”, “rb”);
if($file) {
    $contents = fread($file, 20);
    echo $contents;
}
```

Determining the file end

- `feof($file_pointer)`
  - Returns true if the internal file pointer has reached the end
- `filesize($filename)`
  - Returns the size of the file or `false` on error
  - Results of this function are cached!

```
// Example – print the entire file contents
$file = fopen(“text.txt”, “rb”);
while(!feof($file)) {
    echo fread($file, 32);
}
```

Writing to a file

- Write the specified string to a file:

```
fwrite($resource, $string, $length)
```

  - Works in a binary-safe manner

- **`$resource`** – a pointer to a file
- **`$string`** – the contents to write
- **`$length`** – optional
  - the number of bytes to write

Writing to a file (2)

- Returns:
  - On success, the number of bytes written
  - On failure, **`false`**
- Note that the file must be open in a proper mode for writing
  - Otherwise `fwrite()` will return **`false`**

```
// Example: clear a file and write “Hello World” to it
$file = fopen(“hello.txt”, “w”);
if($file) {
    fwrite($file, “Hello World”);
}
```

Closing a file

- Close a file, resource or a socket connection:

```
fclose($resource)
```

- **`$resource`** – a pointer to the resource to close
- Returns:
  - On success, **`true`**
  - On failure, **`false`**
- Clears all locks to the resource
- Changes made to a file may not be saved until it is closed

SOFTWARE UNIVERSITY FOUNDATION

Basic Operations on Files

Live Demo

Reading entire file at once

- Read an entire stream:

```
stream_get_contents($resource)
```

- Read an entire file:

```
file_get_contents($filename, $file_include_path,
        $context, $offset, $max_length)
```

- **`$filename, $file_include_path, $context`**
  - Like fopen() parameters

Reading entire file at once (2)

- **`$offset`**
  - Specifies the start point of reading
- **`$max_length`**
  - Specifies the maximum number of bytes to read
- Returns:
  - On success, the read content as string
  - On error, **`false`**

```
// Example: display the contents of the file “text.txt”
echo file_get_contents(“text.txt”);
```

Writing a string at once

- Open a file, write the data, and close it:

```
file_put_contents($filename, $data, $flags, $context)
```

- **`$filename`, `$context`**
  - Like fopen() parameters
- **`$data`**
  - The string to write

Writing a string at once (2)

- **`$flags`** – one or more of the following:
  - `FILE_USE_INCLUDE_PATH`
  - `FILE_APPEND`
  - `LOCK_EX`
  - Combining flags – bitwise OR
- Returns:
  - On success, the number of bytes written
  - On failure, **`false`**

```
file_put_contents("hello.txt", "Hello World")
```

Accessing URL resources

- We can download files given as URLs with the same functions:

```
// Example: Read and display the contents of the SoftUni site
<?php
    $file = fopen("http://softuni.bg", "r");
    $content = "";
    while(!feof($file)) {
        $content .= fread($file, 512);
    }

    fclose($file);
    echo $content;
?>
```

More reading and writing

- PHP also has the C-style functions:
  - **`fscanf`**, **`fprintf`**
    - Read / write data parsed according to a format
  - **`fgets`**, **`fputs`**
    - Read / write entire line from / to file
  - **`fgetc`**, **`fputc`**
    - Read / write single character from / to file

More reading and writing (2)

- PHP also has the C-style functions:
  - `fseek`, `ftell`
    - Get / set the position of the internal file pointer
  - `rewind`
    - Set the file pointer at the beginning

SOFTWARE UNIVERSITY FOUNDATION

Advanced Operations on Files

Live Demo

SOFTWARE UNIVERSITY FOUNDATION

Directories

Working with the File System

Listing files in a directory

- List items in a directory according to a pattern:

```
glob($pattern, $flags)
```

- **$pattern**
  - Specified in a OS-dependent way
- **$flags** – some of the flags that apply are:
  - GLOB_ONLYDIR – list only directories
  - GLOB_NOSORT – return the items without sorting
  - GLOB_NOESCAPE – backslashes in pattern are not considered esaping characters

Listing files in a directory (2)

SOFTWARE UNIVERSITY FOUNDATION

- Returns
  - On success, an array representing the items in the directory
  - On failure, **`false`**

```
foreach(glob(“*.txt”) as $file) {
    echo $file . “: ” . filesize($file);
}
```

Listing files in a directory (3)

- A safer alternative to **`glob`** – **`opendir`** / **`readdir`** / **`closedir`**
  - **`glob`** may allow shell injection
- Open directory handle to be used in subsequent **`readdir()`** calls:

```
opendir($path, $context)
```

- Read next item in the directory:

```
readdir($dir_handle)
```

- Close the directory handle:

```
closedir($dir_handle)
```

Creating a directory

- Create a new directory:

```
mkdir($path, $mode, $recursive, $context)
```

- **$path**
  - The new directory path and name
- **$mode**
  - Specify the directory permissions (0777 by default)
  - Ignored on Windows
- **$recursive**
  - If true, create all subdirectories if they do not exist yet

File system functions

- Check if a file / directory exists:

```
mkdir($path, $mode, $recursive, $context)
```

- Copy file / directory

```
copy($source, $destination)
```

- Rename file / directory (also used for moving):

```
rename($old_name, $new_name)
```

- Delete file:

```
unlink($filename)
```

- Delete directory:

```
rmdir($dirname)
```

Names handling

- Given the full path, get only the file name portion (without the directory parts):

```
basename($filename)
```

- Given the full path, get only the directory parts (without the file name):

```
dirname($filename)
```

```
// Example: Show the file and directories of a given path
$path = “home/avatar/file.txt”;
echo basename($path); // temp.txt
echo dirname($path); // home/avatar
```

SOFTWARE UNIVERSITY FOUNDATION

Directories and File System

Live Demo

SOFTWARE UNIVERSITY FOUNDATION

Permissions

Reading, writing, and Executing Rights

Permissions

- Linux permissions are defined at three levels – owner, group, and others
  - Each folder / file has owner and group IDs
  - Read, write and execute rights are assigned for owner, group and other users separately
  - In binary the rights are presented as 3 by 3 bits
    - Usual decimal representation is set of 3 numbers from 0 to 7 (octal representation)
    - Example: 777 means full rights for everyone
    - Example: 760 means full rights for owner, read and write for group members, no access for the rest

Read permissions

- Return the group / owner ID of the file / directory:

```
filegroup($filename)
```

```
fileowner($filename)
```

- Get all permissions of the file / directory as an integer:

```
fileperms($filename)
```

  - Including system file type – socket, link, directory, pipe, etc.

Modify permissions

- Change the group / owner ID of the file / directory:

```
chgrp($filename, $group)
```

```
chown($filename, $user)
```

- Change the mode (permissions) of a file / directory:

```
chmod($filename, $mode)
```

  - **$mode** is specified as integer
  - To ensure proper work, **$mode** should be given in octal mode (e. g. 0755)

Simple permission checks

- Check if PHP can access a file for reading / writing:

```
is_readable($filename)
```

```
is_writeable($filename)
```

  - PHP scripts are usually executed with the apache user and interit its rights, but not always
  - These functions check if the user that executes the script has access to the specified file

Permissions

Live Demo

SOFTWARE UNIVERSITY FOUNDATION

Accessing External Processes

Executing external commands through PHP

External processes

- PHP can start and communicate with other processes
  - Can start a process, send to its input stream and read output / error data
  - The process inherits the permissions of the PHP / Apache process, unless otherwise specified
  - Common use is sending email or processing uploaded files

Execute a program

- Start a process and wait for it to finish:

```
exec($command, $output, $return)
```

- Returns the last line of output
- **`$output`** – optional
  - Array that receives all output data
- **`$return`** – optional
  - Gets the return code of the process

```
$list = array();
exec("ls -l /", $list);
```

Execute a program through the shell

- Execute a program through the shell:

```
shell_exec($command)
```

- Returns the entire output
- Like **exec**, waits for the program to finish

```
$lines = shell_exec("set");
foreach ($lines as $line) {
    if(preg_match("/BASH=(.*)/", $line, $chunk)) {
        echo $chunk[1];
    }
}
```

Starting processes and attaching pipes

- Pipes are streams – one program writes on one end, another reads at the other
  - FIFO (First In, First Out)
  - Each process has at least three pipes – input, output and error
    - Process reads from input, writes at output and error
  - PHP can start process and define handlers for its pipes
    - Can define more pipes if needed

Open a process

- Start a process with given pipe descriptors, working directory, and other details:

```
proc_open($command, $descriptors, $pipes, $dir, $environment, $other)
```

- **`$dir`** – optional
  - Sets the current directory for the process
- **`$environment`** – array, optional
  - Environment variables to be passed
- **`$other`**
  - Other options, mainly for Windows

Open a process (2)

- **`$descriptors`** – indexed array of arrays
  - 0 is input (stdin), 1 is output (stdout), 2 is error (stderr)
  - For each must be specified indexed array
    - The first element is either "file" or "pipe"
    - In case of "pipe", the second element is "r" for reading pipe or "w" for writing pipe
    - In case of "file", the second element is the file, third is "r", "w" or "a" for read, write or append
  - For each of the items in **`$descriptors`**, an element with the same index is created in the **`$pipes`** array
    - The items in **`$pipes`** are used to read / write data

Open a process (3)

- Opening a process – example:

```
$spec = array (
          0 => array (“pipe”, “r”),
          1 => array (“pipe”, “w”),
          2 => array (“file”, “/var/log/mylog.log”, “a”));
$env = array (“PATH” => “/bin/bash”);
$process = proc_open(“/bin/bash”, $spec, $pipes, “/”, $env);
$stdIn = $pipes[0];
$stdOut = $pipes[1];
fwrite($stdIn, “date”);
echo fread($stdOut, 128);
fclose($stdIn);
fclose($stdOut);
proc_close($process);
```

- Pipes can be read and written like ordinary files
- A process is closed with **`proc_close`**

SOFTWARE UNIVERSITY FOUNDATION

Accessing External Processes

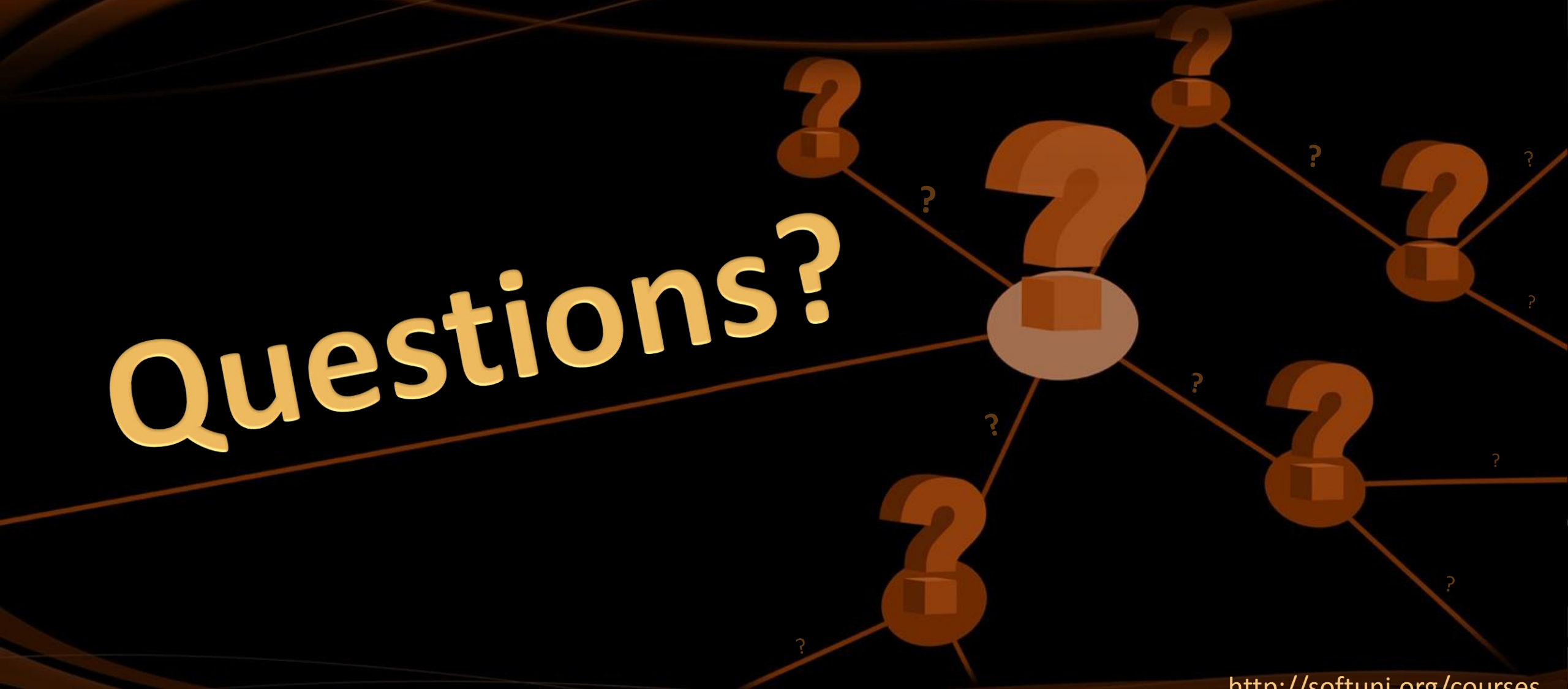
Live Demo

Summary

- PHP provides various functions for working with files and directories
  - Opening
  - Creating
  - Reading / writing
  - Closing
- There are ways to access external processes from the PHP code
  - We can get the result from the process from our code

Working with Files and Processes in PHP

SOFTWARE UNIVERSITY FOUNDATION



License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

- Attribution: this work may contain portions from
  - "Fundamentals of Computer Programming with C#" book by Svetlin Nakov & Co. under CC-BY-SA license
  - "C# Part I" course by Telerik Academy under CC-BY-NC-SA license

Free Trainings @ Software University

- Software University Foundation – softuni.org
- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University @ YouTube
  - youtube.com/SoftwareUniversity
- Software University Forums – forum.softuni.bg