Absolute and Relative Paths in PHP

By W.S. Toh / Tips & Tutorials - PHP

INTRODUCTION

DO YOU KNOW THE WAY?

Welcome to a tutorial on absolute and relative paths in PHP – The difference and when to use each one. You are probably here because you thought you have already set a proper file path, and PHP is still complaining about "missing" files and folders. Yes, it is easy to get lost with the paths in PHP as a beginner, as PHP adopts both absolute and relative pathfinding.

An absolute path refers to defining the full exact file path, for example, "D:\http\project\lib\file.php". While a relative path is based on the current working directory, where the script is located. For example, when we require "html/top.html" in "D:\http\page.php", it will resolve to "D:\http\html\top.html".

Just what the heck is the working directory? Why absolute and relative paths? If you are stuck with this confusion, this guide will help to explain more on how paths work in PHP with examples – Read on to find out!

NAVIGATION

TABLE OF CONTENTS

Microsoft 365
Travaillez là où vous vous sentez inspiré
Gardez le contact avec votre équipe en déplacement et consultez vos fichiers où que us soyez grâce à 1 To de stockage e dans le nuage par

SECTION A

RELATIVE VS ABSOLUTE PATH

So just what is this relative path and absolute path thing that people have been talking about? What is it and how does PHP determine the file paths? This section will answer all of these basic questions.

ABSOLUTE PATH

By defining an absolute path in PHP means giving it the full file path, for example:

D:\http\1a-script.php

```
<?php
require "D:\http\1b-another-script.php";
?>
```

- **The Plus**: With this, you can be sure that it is difficult to mess up the pathfinding in your project.
- **The Minus**: But yep, the downside is that you have to type out the entire file path, and it can get rather tedious with big projects.

RELATIVE PATH

Relative paths, on the other hand, is based on where the current working directory is. For example:

D:\http\1c-script.php

```
<?php
```

- **The Plus**: Saves time, easy to define.
- **The Minus**: Easy to get confused over where the current path is, has a critical pathfinding flaw, which we will explain below.

SECTION B

CURRENT WORKING DIRECTORY

Now that you have a basic understanding of what absolute and relative paths are, this section will dive a little deeper into relative paths – By introducing something called the "current working directory".

WHAT IS THE CURRENT WORKING DIRECTORY?

In the layman terms, the current working directory is simply where the currently executing script is placed in – We can easily get the current working directory with the `getcwd` function.

D:\http\2a-get-cwd.php

```
<?php
// This script is placed in D:\http\
// The current working directory will be D:\http\ when we run this script.
echo getcwd();
?>
```

THE CURRENT WORKING DIRECTORY CHANGES!

So far so good with relative paths? Now, here comes the not-so-straightforward part… The current working directory can change, depending on where the scripts are being placed. Take these 2 scripts for example:

```
<?php
echo getcwd();
require "inside/2c-script.php";
?>
```

D:\http\inside\2c-script.php

```
<?php
echo getcwd();
?>
```

- **If we access** `2b-script.php` – The current working directory in both `2b-script.php` and `2c-script.php` will be D:\http\
- **But if we access** `inside/2c-script.php` – The current working directory in `2c-script.php` will be D:\http\inside\

THE CRITICAL FLAW WITH RELATIVE PATH

Now that you know the current working directory is flexible, you should be able to make a good guess to why it suffers from a critical flaw. Consider the following file structure of a dummy project that will show a list of users:

- `MY-PROJECT` The project root folder.
  - `index.php` The main page of the project.
  - `DOCKET` HTML dockets and components.
    - `users-list.php` Shows a list of users.
  - `LIB` Where we store the library files.
    - `lib-users.php` The user database library file.

D:\MY-PROJECT\index.php

```
<!DOCTYPE html>
<html>
  <body>
  <?php require "DOCKET/users-list.php"; ?>
  </body>
</html>
```

D:\MY-PROJECT\DOCKET\users-list.php

```
<?php
require "LIB/users.php";
foreach ($users as $u) {
  printf("<div>%s</div>", $u['user_name']);
}
?>
```

D:\MY-PROJECT\LIB\lib-users.php

```
<?php
```

```
    PDO::ATTR_EMULATE_PREPARES => false,
  ]
);
$sql = "SELECT * FROM USERS";
$stmt = $pdo->prepare($sql);
$stmt->execute();
$users = $stmt->fetchAll();
?>
```

- **When we access** `index.php` – The current working directory will be D:\MY-PROJECT\. The relative path of `require "LIB/lib-users.php"` will resolve properly to D:\MY-PROJECT\LIB\lib-users.php and not throw any errors.
- **But when we access** `DOCKET/users-list.php` (maybe directly via AJAX) – The working directory will now be changed to D:\MY-PROJECT\DOCKET. The path of `require "LIB/users.php"` will resolve wrongly to D:\MY-PROJECT\DOCKET\LIB\lib-users.php.

Yep, this is pretty much what happens to most beginners, as they scratch their heads wondering "why is the path not working properly".

CHANGING THE WORKING DIRECTORY

So, how do we fix that relative path problem? Thankfully, we can use the `chdir` function to change the current working directory.

D:\http\inside\2d-script.php

```
<?php
// This script is placed in D:\http\inside\
// The current working directory will be D:\http\inside\ when we access it
echo getcwd() . "<br>";

// This will change the current directory to the parent folder
chdir("../");
// The current working directory is now D:\http\
echo getcwd() . "<br>";
?>
```

X

recommend to always use absolute paths.

SECTION C

MAGIC CONSTANTS

Before I share my recommended solution to deal with paths, there is something called the"magic constants" that you need to know – Which we shall walk through in this section.

MAGIC CONSTANTS – GET THE ABSOLUTE PATH OF CURRENT SCRIPT

As with the examples we have gone through above, you should be aware that the current working directory can go wrong in many ways. This is where the PHP magic constants come in handy, giving you the exact location of each script.

- The magic constant `__FILE__` will be the absolute path and file name of the script.
- The magic constant `__DIR__` will be the absolute path of the script.

D:\http\3a-magic.php

```php
<?php
// This script is placed in D:\http\
echo __FILE__ . "<br>"; // D:\http\3a-magic.php
echo __DIR__ . "<br>"; // D:\http\
?>
```

MAGIC CONSTANTS VS WORKING DIRECTORY

D:\http\3b-outside.php

```php
<?php
echo getcwd() . "<br>";
echo __DIR__ . "<br>";
require "D:\http\inside\index.php";
?>
```

D:\http\inside\3c-inside.php

```php
<?php
echo getcwd() . "<br>";
echo __DIR__ . "<br>";
?>
```

Remember from the above examples:

- Magic constants are based where the scripts are placed in.

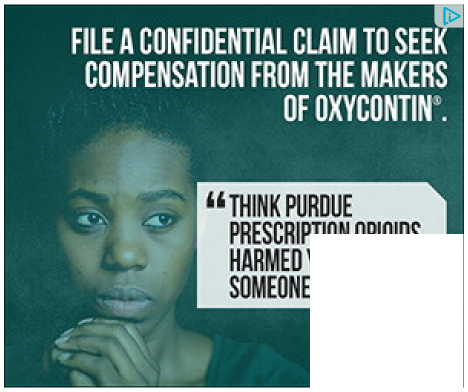
When we access `3b-outside.php`:

- The current working directory will be D:\http\
- The magic directory for `3b-outside.php` will be D:\http\
- But the magic directory for `3c-inside.php` will be D:\http\inside\

When we access `inside/3c-inside.php`:

- The current working directory will be D:\http\inside\
- The magic directory for `3c-inside.php` will be D:\http\inside\

So yes, the magic constants are extremely useful in solving our path issues, and we will go through an example below.


FILE A CONFIDENTIAL CLAIM TO SEEK COMPENSATION FROM THE MAKERS OF OXYCONTIN®.
"THINK PURDUE PRESCRIPTION OPIOIDS HARMED SOMEONE

RECOMMENDATION – ALWAYS USE THE MAGIC CONSTANT

Let us first bring back the problematic duo:

D:\MY-PROJECT\index-bad.php

```
<!DOCTYPE html>
<html>
  <body>
  <?php require "DOCKET/users-list-bad.php"; ?>
  </body>
</html>
```

D:\MY-PROJECT\DOCKET\users-list-bad.php

```
<?php
require "LIB/users.php";
foreach ($users as $u) {
  printf("<div>%s</div>", $u['user_name']);
```

Quick recap – If we directly access users-list-bad.php, the file path for require "LIB/users.php" will resolve wrongly to D:\MY-PROJECT\DOCKET\LIB\users.php. We can use magic constants to fix that.

D:\MY-PROJECT\index-good.php

```
<!DOCTYPE html>
<html>
  <body>
  <?php require __DIR__ . "DOCKET/users-list.php"; ?>
  </body>
</html>
```

D:\MY-PROJECT\DOCKET\users-list-good.php

```
<?php
// dirname(__DIR__) will resolve to the parent folder
// Which is D:\MY-PROJECT\
require dirname(__DIR__) . "lib/users.php";
foreach ($users as $u) {
  printf("<div>%s</div>", $u['user_name']);
}
?>
```

Since the magic constants are “anchored” to the folder of the scripts themselves, the paths will always be correct regardless of where you access them from.

SECTION D

MORE PATH YOGA

At this point, you should already have sufficient ninja skills to deal with the paths. But that is not the end to it… There are a lot more to path yoga, and we will walk through them in this section.

PATH RELATED SERVER VARIABLES

Apart from the magic constants, there are also a few useful stuff in the $_SERVER variable, although I personally don’t use these often.

- **DOCUMENT_ROOT** – Contains the root HTTP folder.
- **PHP_SELF** – Contains the relative path to the script.
- **SCRIPT_FILENAME** – Contains the full path to the script.

D:\http\4a-server-var.php

```
<?php
```

```
?>
```

PATH INFO

The PHP `pathinfo` function quickly gives you bearings on a given path:

- **dirname** – The directory of the given path.
- **basename** – The filename with extension.
- **filename** – Filename, without extension.
- **extension** – File extension.

D:\http\4b-path-info.php

```
<?php
$parts = pathinfo("D:\http\inside\index.php");

echo $parts['dirname'] . "<br>"; // D:\http\test\inside
echo $parts['basename'] . "<br>"; // index.php
echo $parts['filename'] . "<br>"; // index
echo $parts['extension'] . "<br>"; // php
?>
```

PHP BASENAME

The `basename` function will give you the trailing directory or file of a given path.

D:\http\4c-basename.php

```
<?php
// DIRECTORY
$path = "D:\http\test";
echo basename($path); // test

// FILE
$path = "D:\http\test\index.php";
echo basename($path); // index.php

// YOU CAN SPECIFY A SUFFIX TO CUTOFF
$path = "D:\http\test\index.php";
echo basename($path, ".php"); // index
?>
```

PHP DIRNAME

```
$path = "D:\http\test";
echo dirname($path); // D:\http\

// YOU CAN SPECIFY THE NUMBER OF LEVELS TO GO UP
echo dirname($path,2); // D:\
?>
```

PHP REALPATH

The `realpath` function gives you a canonicalized absolute path. Useful when working with relative paths (with the current working directory).

D:\http\index.php

```
<?php
echo realpath("inside") . "<br>"; // D:\http\inside
echo realpath("../") . "<br>"; // D:\http

// OF COURSE, IF YOU FEED IT ABSOLUTE PATH... IT WILL GIVE YOU A CLEAN ABSOLUTE PATH
echo realpath("C:/Windows/"); // C:\Windows
?>
```

FORWARD OR BACKWARD SLASH!?

Windows uses \ and Linux uses /… So how do we please both of them? Actually, modern Windows will accept both and we don’t have to worry too much. But for safety, we can always use the DIRECTORY_SEPARATOR constant.

```
$path = "folder" . DIRECTORY_SEPARATOR . "inside" . DIRECTORY_SEPARATOR . "file.txt";
```

X

SECTION E

SUMMARY

That's the end of the tutorial, and yes, that is a lot of stuff to digest… So here is a summary of all the path-related variables, functions, and a cheat sheet.

ALL THE PATH-RELATED VARIABLES

Variable	**Description**
__FILE__	The absolute path and file name of the current script.
__DIR__	The absolute path of the current script.
$_SERVER["DOCUMENT_ROOT"]	The root HTTP folder.
$_SERVER["PHP_SELF"]	Relative path to the script.
$_SERVER["SCRIPT_FILENAME"]	Full path (and file name) to the script.
DIRECTORY_SEPARATOR	This is automatically a forward or backward slash, depending on the system.

ALL THE PATH-RELATED FUNCTIONS

Function	**Description**
getcwd	Returns the current working directory.
chdir	Changes the current working directory.
pathinfo	Gives you the path information of a given path. • **dirname** – The directory of the given path. • **basename** – The filename with extension. • **filename** – Filename, without extension. • **extension** – File extension.
basename	Gives you the trailing directory or file of a given path.
dirname	Gives you the parent directory of a given path.

CHEAT SHEET

PHP Path Functions & Variables
(click to enlarge)

LINKS & REFERENCES

- PHP Magic Constants
- getcwd
- chdir
- dirname
- basename
- pathinfo
- realpath

CLOSING

PHP PATH YOGA

Thank you for reading, and we have come to the end of this guide. I hope it has helped you find the true path, and if you have anything to add to this guide, please feel free to comment below. Good luck and happy coding!

W.S. Toh

W.S. Toh is a senior web developer and SEO practitioner with over 15 years of experience in building websites and systems. When he is not secretly being an evil tech ninja, he enjoys doing photography and working on DIY projects.

← Previous Post

Next Post →

2 thoughts on “Absolute and Relative Paths in PHP”

PATRICE
JANUARY 16, 2020 AT 5:08 AM

You wrote “Windows uses \ and Linux uses /”
Not exactly… Only Windows is using \ and everyone else use /
It came from their first DOS, which did not have a directory/folder system, and after a while when they updated their DOS they were not able to put the standard / as their customers using their DOS had already used the / for something else, so Microsoft adopted the backslash as an alternative as it was recognizable to the /

Reply X

W.S. TOH
JANUARY 16, 2020 AT 5:25 AM

Eh… Thank you for rephrasing “Windows uses \ and Linux uses /” to “Only Windows is using \ and everyone else use /”?

Sorry, I am just very confused about what you are trying to share here. Maybe a piece of history to why Mircosoft is the odd one out? If that is so, then… Thank you for the history lesson??

Reply

Leave a Comment

Your email address will not be published. Required fields are marked *

Type here..

Name*

Email*

Post Comment »

X

Recent Posts

2 Steps Simple Responsive Pure CSS Hamburger Menu

3 Ways To Redirect A Webpage In Javascript

4 Ways to Encrypt Decrypt and Verify Passwords in PHP

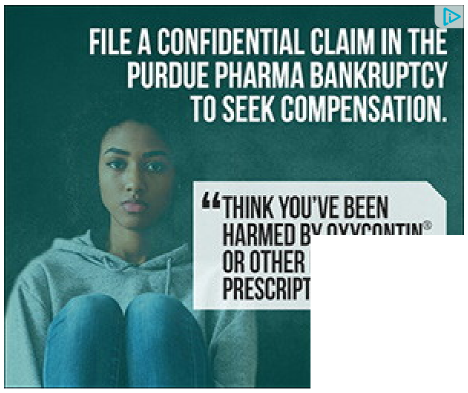
How to Create a Simple Minesweeper Game With Vanilla Javascript

4 Ways to Detect Browser With Javascript

Follow Us

Pinterest

YouTube

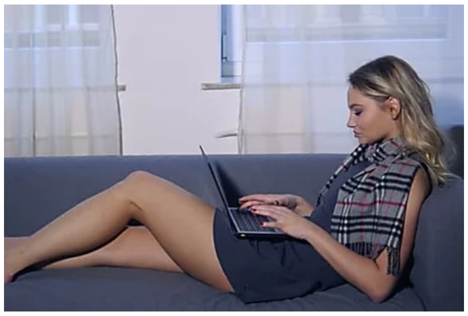

FILE A CONFIDENTIAL CLAIM IN THE PURDUE PHARMA BANKRUPTCY TO SEEK COMPENSATION.
"THINK YOU'VE BEEN HARMED BY OXYCONTIN® OR OTHER PRESCRIPT

X

Search

Search …

Sponsored Content



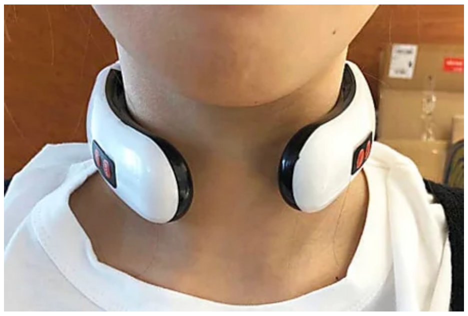
Mac User Warning: The Only Providers You Should Trust To Protect Your Mac.

My Antivirus Review

Chiropractors Baffled: Simple Stretch Relieves Years of Back Pain (Watch)

healthbenefits.live

X



This Neck Relief Device Might Be The Most Amazing Invention in 2020

NeckRelax

[Photos] This '70s Crockery Is Worth A Fortune

Graduatez

Recommended by

ezoic report this ad