Absolute & Relative Paths In PHP – A Simple Guide

By W.S. Toh / Tips & Tutorials - PHP / October 31, 2021

Welcome to a quick tutorial on absolute and relative paths in PHP. So you have set a verified file path, but PHP is still complaining about "missing" files and folders? Yes, it is easy to get lost with absolute and relative paths in PHP.

- 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`.

The covers the basics, but let us walk through more on how file paths work in PHP – Read on!

ⓘ I have included a zip file with all the source code at the start of this tutorial, so you don't have to copy-paste everything… Or if you just want to dive straight in.

EXAMPLE CODE DOWNLOAD

Click here to download all the example source code, I have released it under the MIT license, so feel free to build on top of it or use it in your own project.

PHP FILE PATH

All right, let us now get into the examples of how file paths work in PHP.

1) ABSOLUTE & RELATIVE PATH

D:\http\1a-dummy.php

```
<?php
echo "Foo Bar";
```

D:\http\1b-abs-rel.php

```
// (A) ABSOLUTE FILE PATH (FULL PATH)
require "D:\http\1a-dummy.php";

// (B) RELATIVE PATH (TO CURRENT WORKING DIRECTORY)
require "1a-dummy.php";
```

As in the introduction above:

- An absolute file path is simply the "full file path". Although a bit long-winded, it's hard to mess up with this one.
- On the other hand, the relative file path is based on the current working directory.

But just what the heck is the "current working directory"? Follow up with the next example.

2) CURRENT WORKING DIRECTORY

D:\http\2-cwd.php

```
<?php
// IF THIS SCRIPT IS PLACED IN D:\HTTP
// CURRENT WORKING DIRECTORY WILL BE D:\HTTP
echo getcwd();
```

In simple terms, the current working directory is the folder where the script is placed in. We can easily get the current working directory with the `getcwd()` function.

3) CONFUSION WITH THE CURRENT WORKING DIRECTORY

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

```
<?php
require "D:\http\2-cwd.php";
```

So far so good with relative paths? Now comes the part that destroyed many beginners, take extra note that this example script is placed at `D:\http\inside`. Guess what `getcwd()` will show when we run this example?

Yes, the current working directory now changes to `D:\http\inside`. Take extra note, the current working directory is fixed to the first script that runs.

4) CHANGING THE WORKING DIRECTORY

D:\http\inside\4-change-cwd.php

```
<?php
// (A) SHOW CURRENT WORKING DIRECTORY
require "D:\http\2-cwd.php"; // D:\HTTP\INSIDE

// (B) CHANGE TO PARENT FOLDER
chdir("../");
echo getcwd(); // D:\HTTP
```

The current working directory will surely mess up a lot of relative paths. So, how do we fix this problem? Thankfully, we can use the `chdir()` function to change the current working directory.

5) MAGIC CONSTANTS

D:\http\5-magic.php

```
<?php
// (A) CURRENT FILE - D:\HTTP\5-MAGIC.PHP
echo __FILE__ . "<br>";

// (B) CURRENT FOLDER - D:\HTTP
echo __DIR__ . "<br>";
```

Even if we are able to change the working directory, relative paths are still messy in big projects. I will recommend using magic constants instead.

- `__FILE__` is the full path and file name where the current script is located.
- `__DIR__` is the folder where the current script is located.

6) MAGIC CONSTANTS VS WORKING DIRECTORY

D:\http\inside\6-magic.php

```
<?php
// (A) CURRENT FILE - D:\HTTP\INSIDE\6-MAGIC.PHP
echo __FILE__ . "<br>";

// (B) CURRENT FOLDER - D:\HTTP\INSIDE
echo __DIR__ . "<br>";

// (C) GET PARENT FOLDER
$parent = dirname(__DIR__) . DIRECTORY_SEPARATOR;
require $parent . "5-magic.php";
```

To lessen the confusion, take note that the magic constants are different from the current working directory.

- The current working directory is based on the first script that we run.
- Magic constants are based on where the scripts are placed in.

Yes, magic constants are the better way to do “pathfinding”, and to build the absolute paths.

7) MORE PATH YOGA

D:\http\7-extra.php

```
<?php
// (A) SERVER PATH VARIABLES
echo $_SERVER["DOCUMENT_ROOT"] . "<br>"; // D:/HTTP
echo $_SERVER["PHP_SELF"] . "<br>"; // 7-EXTRA.PHP
echo $_SERVER["SCRIPT_FILENAME"] . "<br>"; // D:/HTTP/7-EXTRA.PHP

// (B) PATH INFO
$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

// (C) BASENAME
$path = "D:\http\inside";
echo basename($path) . "<br>"; // INSIDE
$path = "D:\http\inside\foo.php";
echo basename($path) . "<br>"; // FOO.PHP
$path = "D:\http\inside\foo.php";
echo basename($path, ".php") . "<br>"; // FOO

// (D) DIRNAME
$path = "D:\\http\\inside\\";
echo dirname($path) . "<br>"; // D:\HTTP
echo dirname($path, 2) . "<br>"; // D:\
```

```
// (E) REALPATH
echo realpath("") . "<br>"; // D:\HTTP
echo realpath("../") . "<br>"; // D:\
```

Finally, this is a quick crash course on the various variables and functions that may help you find the file path.

SERVER SUPERGLOBAL

- `$_SERVER["DOCUMENT_ROOT"]` – Contains the root HTTP folder.
- `$_SERVER["PHP_SELF"]` – The relative path to the script.
- `$_SERVER["SCRIPT_FILENAME"]` – The full path to the script.

PATH INFO

The PHP `pathinfo()` function will give 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.

BASENAME

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

DIRNAME

The `dirname()` function will give you the parent directory of a given path.

REALPATH

The `realpath()` function gives you a canonicalized absolute path… Kind of useful, but still based on the current working directory.

EXTRA) FORWARD OR BACKWARD SLASH?

In case you have not realized:

- Windows use \
- Linux and Mac uses /

Not really a big problem though, just use `DIRECTORY_SEPARATOR` in PHP and it will automatically resolve to the “correct slash”.

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 filename) 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()	Get the trailing directory or file of a given path.
dirname()	Get the parent directory of a given path.
realpath()	Canonicalized absolute path.

LINKS & REFERENCES

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

https://code-boxx.com/php-absolute-relative-path/

15

you should use a config file that will be included in each file first line, for example your app look like this

root / App / Plugins

inside your root dir : `app-config.php`

```
if ( !defined('ABSPATH') )
    define('ABSPATH', dirname(__FILE__) . '/');
```

now, suppose you have to include a plugin file, so

inside your Plugin dir : `my-plugin.php`

```
require_once '../../app-config.php';
```

now everything below this line can use ABSPATH

example do you want to load an image

```
<img src='".ABSPATH."Public/images/demo.png' alt=''/>
```

now, the thing is more simple if your app is designed to automatically load some files like

```
plugin-widget-1.php
```

so that everything inside this file or any other file loaded by the `my-plugin.php` file can use the ABSPATH without include each time the `app-config.php` file.

with this in mind you can have all the short-hand you want into the `app-config.php` example

```
define('UPLOAD_PATH', ABSPATH. 'Public/uploads/');
define('IMAGES_PATH', ABSPATH. 'Public/images/');
define('HELPERS_PATH', ABSPATH. 'App/helpers/');
...
```

so, now that you have all defined, if you need to move a file, let's say one folder forward example:

root / App / Plugins / Utils

just inlucde `require_once '../../../app-config.php';`

obviously i suppose that you are not changing paths each time =) anyway if you need to do so is always more simple to change one file inclusion instead of hundreds.

hope this make sense to you =)

1

I always use absolute paths, but I also start any custom PHP project with a bootstrap file where I define the most commonly used paths as constants, based on values extracted from $_SERVER.

This is how I define my root paths :

```
define("LOCAL_PATH_ROOT", $_SERVER["DOCUMENT_ROOT"]);
define("HTTP_PATH_ROOT", isset($_SERVER["HTTP_HOST"]) ? $_SERVER["HTTP_HOST"] :
(isset($_SERVER["SERVER_NAME"]) ? $_SERVER["SERVER_NAME"] : '_UNKNOWN_'));
```

The path LOCAL_PATH_ROOT is the document root. The path HTTP_PATH_ROOT is the equivalent when accessing the same path via HTTP.

At that point, converting any local path to an HTTP path can be done with the following code :

```
str_replace(LOCAL_PATH_ROOT, RELATIVE_PATH_ROOT, $my_path)
```

If you want to ensure compatibility with Windows based servers, you'll need to replace the directory seperator with a URL seperator as well :

```
str_replace(LOCAL_PATH_ROOT, RELATIVE_PATH_ROOT,
str_replace(DIRECTORY_SEPARATOR, '/', $my_path))
```

Here's the full bootstrap code that I'm using for the [PHP PowerTools boilerplate] :

```
defined('LOCAL_PATH_BOOTSTRAP') || define("LOCAL_PATH_BOOTSTRAP", __DIR__);

// ----------------------------------------------------------------------
// DEFINE SEPERATOR ALIASES
// ----------------------------------------------------------------------
define("URL_SEPARATOR", '/');
define("DS", DIRECTORY_SEPARATOR);
define("PS", PATH_SEPARATOR);
define("US", URL_SEPARATOR);

// ----------------------------------------------------------------------
// DEFINE ROOT PATHS
// ----------------------------------------------------------------------
define("RELATIVE_PATH_ROOT", '');
define("LOCAL_PATH_ROOT", $_SERVER["DOCUMENT_ROOT"]);
define("HTTP_PATH_ROOT",
        isset($_SERVER["HTTP_HOST"]) ?
        $_SERVER["HTTP_HOST"] : (
        isset($_SERVER["SERVER_NAME"]) ?
        $_SERVER["SERVER_NAME"] : '_UNKNOWN_'));

// ----------------------------------------------------------------------
// DEFINE RELATIVE PATHS
// ----------------------------------------------------------------------
define("RELATIVE_PATH_BASE",
        str_replace(LOCAL_PATH_ROOT, RELATIVE_PATH_ROOT, getcwd()));
define("RELATIVE_PATH_APP", dirname(RELATIVE_PATH_BASE));
define("RELATIVE_PATH_LIBRARY", RELATIVE_PATH_APP . DS . 'vendor');
define("RELATIVE_PATH_HELPERS", RELATIVE_PATH_BASE);
define("RELATIVE_PATH_TEMPLATE", RELATIVE_PATH_BASE . DS . 'templates');
define("RELATIVE_PATH_CONFIG", RELATIVE_PATH_BASE . DS . 'config');
define("RELATIVE_PATH_PAGES", RELATIVE_PATH_BASE . DS . 'pages');
define("RELATIVE_PATH_ASSET", RELATIVE_PATH_BASE . DS . 'assets');
define("RELATIVE_PATH_ASSET_IMG", RELATIVE_PATH_ASSET . DS . 'img');
define("RELATIVE_PATH_ASSET_CSS", RELATIVE_PATH_ASSET . DS . 'css');
define("RELATIVE_PATH_ASSET_JS", RELATIVE_PATH_ASSET . DS . 'js');

// ----------------------------------------------------------------------
// DEFINE LOCAL PATHS
// ----------------------------------------------------------------------
define("LOCAL_PATH_BASE", LOCAL_PATH_ROOT . RELATIVE_PATH_BASE);
define("LOCAL_PATH_APP", LOCAL_PATH_ROOT . RELATIVE_PATH_APP);
define("LOCAL_PATH_LIBRARY", LOCAL_PATH_ROOT . RELATIVE_PATH_LIBRARY);
define("LOCAL_PATH_HELPERS", LOCAL_PATH_ROOT . RELATIVE_PATH_HELPERS);
define("LOCAL_PATH_TEMPLATE", LOCAL_PATH_ROOT . RELATIVE_PATH_TEMPLATE);
define("LOCAL_PATH_CONFIG", LOCAL_PATH_ROOT . RELATIVE_PATH_CONFIG);
define("LOCAL_PATH_PAGES", LOCAL_PATH_ROOT . RELATIVE_PATH_PAGES);
```

```
define("LOCAL_PATH_ASSET", LOCAL_PATH_ROOT . RELATIVE_PATH_ASSET);
define("LOCAL_PATH_ASSET_IMG", LOCAL_PATH_ROOT . RELATIVE_PATH_ASSET_IMG);
define("LOCAL_PATH_ASSET_CSS", LOCAL_PATH_ROOT . RELATIVE_PATH_ASSET_CSS);
define("LOCAL_PATH_ASSET_JS", LOCAL_PATH_ROOT . RELATIVE_PATH_ASSET_JS);

// ----------------------------------------------------------------------
// DEFINE URL PATHS
// ----------------------------------------------------------------------
if (US === DS) { // needed for compatibility with windows
    define("HTTP_PATH_BASE", HTTP_PATH_ROOT . RELATIVE_PATH_BASE);
    define("HTTP_PATH_APP", HTTP_PATH_ROOT . RELATIVE_PATH_APP);
    define("HTTP_PATH_LIBRARY", false);
    define("HTTP_PATH_HELPERS", false);
    define("HTTP_PATH_TEMPLATE", false);
    define("HTTP_PATH_CONFIG", false);
    define("HTTP_PATH_PAGES", false);
    define("HTTP_PATH_ASSET", HTTP_PATH_ROOT . RELATIVE_PATH_ASSET);
    define("HTTP_PATH_ASSET_IMG", HTTP_PATH_ROOT . RELATIVE_PATH_ASSET_IMG);
    define("HTTP_PATH_ASSET_CSS", HTTP_PATH_ROOT . RELATIVE_PATH_ASSET_CSS);
    define("HTTP_PATH_ASSET_JS", HTTP_PATH_ROOT . RELATIVE_PATH_ASSET_JS);
} else {
    define("HTTP_PATH_BASE", HTTP_PATH_ROOT .
            str_replace(DS, US, RELATIVE_PATH_BASE));
    define("HTTP_PATH_APP", HTTP_PATH_ROOT .
            str_replace(DS, US, RELATIVE_PATH_APP));
    define("HTTP_PATH_LIBRARY", false);
    define("HTTP_PATH_HELPERS", false);
    define("HTTP_PATH_TEMPLATE", false);
    define("HTTP_PATH_CONFIG", false);
    define("HTTP_PATH_PAGES", false);
    define("HTTP_PATH_ASSET", HTTP_PATH_ROOT .
            str_replace(DS, US, RELATIVE_PATH_ASSET));
    define("HTTP_PATH_ASSET_IMG", HTTP_PATH_ROOT .
            str_replace(DS, US, RELATIVE_PATH_ASSET_IMG));
    define("HTTP_PATH_ASSET_CSS", HTTP_PATH_ROOT .
            str_replace(DS, US, RELATIVE_PATH_ASSET_CSS));
    define("HTTP_PATH_ASSET_JS", HTTP_PATH_ROOT .
            str_replace(DS, US, RELATIVE_PATH_ASSET_JS));
}

// ----------------------------------------------------------------------
// DEFINE REQUEST PARAMETERS
// ----------------------------------------------------------------------
define("REQUEST_QUERY",
        isset($_SERVER["QUERY_STRING"]) && $_SERVER["QUERY_STRING"] != '' ?
        $_SERVER["QUERY_STRING"] : false);
define("REQUEST_METHOD",
        isset($_SERVER["REQUEST_METHOD"]) ?
        strtoupper($_SERVER["REQUEST_METHOD"]) : false);
define("REQUEST_STATUS",
        isset($_SERVER["REDIRECT_STATUS"]) ?
        $_SERVER["REDIRECT_STATUS"] : false);
define("REQUEST_PROTOCOL",
        isset($_SERVER["HTTP_ORIGIN"]) ?
        substr($_SERVER["HTTP_ORIGIN"], 0,
        strpos($_SERVER["HTTP_ORIGIN"], '://') + 3) : 'http://');
define("REQUEST_PATH",
        isset($_SERVER["REQUEST_URI"]) ?
        str_replace(RELATIVE_PATH_BASE, '',
        $_SERVER["REQUEST_URI"]) : '_UNKNOWN_');
define("REQUEST_PATH_STRIP_QUERY",
        REQUEST_QUERY ?
        str_replace('?' . REQUEST_QUERY, '', REQUEST_PATH) : REQUEST_PATH);
```

```
// ------------------------------------------------------------------------
// DEFINE SITE PARAMETERS
// ------------------------------------------------------------------------
define("PRODUCTION", false);
define("PAGE_PATH_DEFAULT", US . 'index');
define("PAGE_PATH",
        (REQUEST_PATH_STRIP_QUERY === US) ?
        PAGE_PATH_DEFAULT : REQUEST_PATH_STRIP_QUERY);
```

If you add the above code to your own project, outputting all user constants at this point (which can do with `get_defined_constants(true)` should give a result that looks somewhat like this :

```
array (
  'LOCAL_PATH_BOOTSTRAP' => '/var/www/libraries/backend/Data/examples',
  'URL_SEPARATOR' => '/',
  'DS' => '/',
  'PS' => ':',
  'US' => '/',
  'RELATIVE_PATH_ROOT' => '',
  'LOCAL_PATH_ROOT' => '/var/www',
  'HTTP_PATH_ROOT' => 'localhost:8888',
  'RELATIVE_PATH_BASE' => '/libraries/backend/Data/examples',
  'RELATIVE_PATH_APP' => '/libraries/backend/Data',
  'RELATIVE_PATH_LIBRARY' => '/libraries/backend/Data/vendor',
  'RELATIVE_PATH_HELPERS' => '/libraries/backend/Data/examples',
  'RELATIVE_PATH_TEMPLATE' => '/libraries/backend/Data/examples/templates',
  'RELATIVE_PATH_CONFIG' => '/libraries/backend/Data/examples/config',
  'RELATIVE_PATH_PAGES' => '/libraries/backend/Data/examples/pages',
  'RELATIVE_PATH_ASSET' => '/libraries/backend/Data/examples/assets',
  'RELATIVE_PATH_ASSET_IMG' => '/libraries/backend/Data/examples/assets/img',
  'RELATIVE_PATH_ASSET_CSS' => '/libraries/backend/Data/examples/assets/css',
  'RELATIVE_PATH_ASSET_JS' => '/libraries/backend/Data/examples/assets/js',
  'LOCAL_PATH_BASE' => '/var/www/libraries/backend/Data/examples',
  'LOCAL_PATH_APP' => '/var/www/libraries/backend/Data',
  'LOCAL_PATH_LIBRARY' => '/var/www/libraries/backend/Data/vendor',
  'LOCAL_PATH_HELPERS' => '/var/www/libraries/backend/Data/examples',
  'LOCAL_PATH_TEMPLATE' => '/var/www/libraries/backend/Data/examples/templates',
  'LOCAL_PATH_CONFIG' => '/var/www/libraries/backend/Data/examples/config',
  'LOCAL_PATH_PAGES' => '/var/www/libraries/backend/Data/examples/pages',
  'LOCAL_PATH_ASSET' => '/var/www/libraries/backend/Data/examples/assets',
  'LOCAL_PATH_ASSET_IMG' =>
'/var/www/libraries/backend/Data/examples/assets/img',
  'LOCAL_PATH_ASSET_CSS' =>
'/var/www/libraries/backend/Data/examples/assets/css',
  'LOCAL_PATH_ASSET_JS' => '/var/www/libraries/backend/Data/examples/assets/js',
  'HTTP_PATH_BASE' => 'localhost:8888/libraries/backend/Data/examples',
  'HTTP_PATH_APP' => 'localhost:8888/libraries/backend/Data',
  'HTTP_PATH_LIBRARY' => false,
```

https://stackoverflow.com/questions/8356547/absolute-vs-relative-paths

Absolute paths are better from a performance point of view when using an opcode cache or lots of require/include statement (although its only noticeable when you start to include hundreds of files, as might happen when using a framework like Zend/Symfony/etc).

With a relative path the opcode cache and php must work out the files realpath each time before it can work out if it already knows about the file and if it needs to load it again. PHP internally maintains a hashmap of files to file locations which is very quick as long it doesn't have to do the above calculation each time.

Relative and absolute paths, in the file system and on the web server.

Intro

Your site exists in two realms at once: the real and the virtual one.

For the site visitors it's entirely a **virtual** server, which in many ways is different from a real one. *There are no files for starter*. I know, it's hard to believe at first, but it's a fact. In the address like `http://example.com/file.html`, `file.html` is not a file. It's a part of URI, a virtual resource. There could be or could be not a real file with such a name, but it doesn't matter. Your browser cannot know that, and don't need to. All it needs to know is an address.

For the site developer, on the other hand, their site is a certain program running on a particular server, on the very real computer with HDD, files and directories. And your PHP script, while reading data files or including other scripts, is working with such real files that exist on the *physical medium*.

So this dualism is the root of many problems.

PHP users confuse these matters badly at first, doing things like being unable to locate an existing file, confusing hyperlinks with files, including local files via HTTP and such.

However, to sort these things out all you need is to grasp just two simple concepts:

1. The difference between **absolute** and **relative** paths.
2. The difference between the **root of the web server** and the **filesystem root**.

The difference between absolute and relative paths

It's fairly simple.

- If the path is built starting from the system **root**, it is called **absolute**.
- If the path is built starting from the current location, it is called **relative** (which makes sense, as it is relative to our present position)

It's exactly the same as with the real life directions. Given the absolute address, a postal one, like "7119 W Sunset Blvd West Hollywood, CA 90046" you can find the location from anywhere.

However, given the relative directions, like "keep three blocks this way and then and turn to the right" would work from the current location only, otherwise sending you astray.

So it goes for the paths in the computer world: given the absolute address, you can always get to the place, no matter from where you started. Whereas relative path is tricky, and should be used with caution, only when you positively know where you are at the moment.

Absolute paths

So again: an absolute path is one starting from the system root

Some absolute path examples:

```
/var/www/site/forum/index.php
/img/frame.gif
C:\windows\command.com
```

Note that in Unix-like systems (and web-servers) the root is defined as a slash - `/`. And this is very important to know. It is not just a marker, but already a full qualified address, a path. Type `cd /` in your unix console and you will get to the root directory. Exactly the same is true for all web servers. So you can tell that in the `http://example.com/` address the trailing slash is not for the decoration but a regular address itself - the address of the home page.

On Windows, the filesystem doesn't have the common root for the whole system but split between disks, so an absolute paths starts from the drive letter. Whereas each disk has its own root, which is a *back*slash - `\`. So you can type `cd \` and get to the root of the current disk.

So you can tell that windows is rather confusing, but for the simplicity we would pretend that we have only one disk, and within its boundaries the rules are pretty much the same as in Unix.

So now you can tell an absolute path from a relative one - it is starting from the root, which is:

- on a Unix file system it's `/`
- on a web serer it's again `/`
- on Windows it's either `\` (for the current disk) or `D:\` (system-wide)

Relative paths

If you don't supply the root, it means that your path is relative.

The simplest example of relative path is just a file name, like `index.html`. So one should be careful with relative paths. If your current directory is `/about/` then `index.html` would be one, but if you switch it to `/contacts/` then it will be another.

Other relative path examples:

- `./file.php` (the file is in the current folder. The same as just `file.php`)
- `images/picture.jpg` (the file is in the images folder that is in the current directory)
- `../file.php` (file is in the folder that is one level higher than the current directory)
- `../../file.php` (file is in the folder that is two levels higher than the current directory)

What you ought to know is that the system, when encountered a relative path, **always builds it up to the absolute one**. Both web-server and file system are doing that but different ways. So, let's learn them.

Document root

This is the most interesting part. There is a point where the real world meets the virtual one.

Imagine there is a file like `/var/www/site/forum/index.php`. While on the web-server its address is `http://www.site.ru/forum/index.php`

And here the point can be clearly seen: there is a part, common for both addresses: `/forum/index.php`, which is the very source of confusion.

For the browser, this path is perfectly absolute, starting from the root of the web-server. Whereas for the script it's only a part of the full path - the filesystem path. And if you try to use it in PHP it will result in a failure: there is no `/forum/` catalog on the *HDD*!

To get the working path to this file, we have to add the missing part. In our example it's `/var/www/site`, which is called `DOCUMENT_ROOT` and is the most important configuration option for the file system interactions. In PHP you can access it via `$_SERVER['DOCUMENT_ROOT']`.

So now you can tell that to make **any** file system path work, it should be absolute and built using `DOCUMENT_ROOT`. So a correct PHP code to access `/forum/index.php` from PHP would be

```
$path = $_SERVER['DOCUMENT_ROOT'] . "/forum/index.php";
```

here we are using web-server part of the path, prepending it with the document root. Voila!

Web server paths

are much simpler.

Like it was said before, for the browser, there are no files on the server. A site user never has an access to the server's file system. For the browser, there is a **site root** only. Which is constant *and always simply a slash*.

Therefore, to make an HTML link absolute, just build it from the site root - and you will never see a 404 error for the existing file again!

Imagine your site has two sections,

```
http://www.example.com/about/info.php
and
http://www.example.com/job/vacancy.php
```

and in the `info.php` you want to link to `vacancy.php`. If you make it as is, `<a href=vacancy.php>`, then browser won't find it! Remember, it always tries to build up the link to the full one, using the current location, which is `/about/` and so the resulting path is `/about/vacancy.php` which is wrong. To make it right, we have to make this link absolute, starting from the site root: `/job/vacancy.php`

So it goes for all the internal links on the site - images, js and css files, hyperlinks or any other resource that can be clicked on or loaded on the page.

For the local resources it's better to make it path only, without protocol and domain - like `/job/vacancy.php`. Whereas for the external resources these attributes are obligatory, and so it should be a fully qualified URL like `http://www.example.com/job/vacancy.php`.

Console scripts. Single entry point

It's a pity, but for the console scripts our useful `$_SERVER['DOCUMENT_ROOT']` variable is unavailable. So we are bound to use paths relative to the calling script, derived from the current script's location.

For example, if your application is hosted in `/var/www/app` and there are two subfolders, `/var/www/app/bin` and `/var/www/app/config`, and you want to access the latter from the former, you can write the following code:

```
$config_path = __DIR__.'/../config/settings.php';
require $config_path;
```

Although technically absolute (starting from a slash), this path is essentially relative to the calling script, because if the calling script will be moved into another directory, it won't find the configuration file anymore.

This is why it is recommended to use a single entry point for your application. Or - as in our case - two entry points, one for web requests and one for console commands.

So for our fictional application we would have three files - an entry point for the web front, an entry point for console applications and a bootstrap file:

- /var/www/app/html/index.php
- /var/www/app/bin/console.php
- /var/www/app/config/bootstrap.php

Then we could write the following code (among other things) in `bootstrap.php`:

```
define('ROOT_DIR', realpath(__DIR__.'/..'));
```

to define the `ROOT_DIR` constant that contains the path to our application's root directory (which is directly above the `config` dir).

And then in both `index.php` and `console.php` the

```
require __DIR__.'/../config/bootstrap.php';
```

to make all the bootstrap stuff available, including the ROOT_DIR constant. From now on we can use it to build absolute paths starting from the root directory (as long as our scripts are called through the entry point either web or console one):

```
include ROOT_DIR.'/config/settings.php';
```

Example implementations can be found in Laravel's Artisan or Symfony Console.

Of course, both entry points should implement a sort of resolver to call all other pages and console scripts but that's slightly out of scope of this article.

Helpful PHP commands and constants

There are may hepful commands and constants in PHP to ease the path interpolation. Some of them are:

- `__FILE__` a constant that contains the full absolute path to the file which is currently executing.
- `__DIR__` a constant that contains the path to a directory where lies the file which is currently executing (effectively it's just `__FILE__` without the filename and the ending slash)
- `realpath()` command will convert a relative path to an absolute one
- `getcwd()` will give you the current directory

Related articles:

- Articles
- MVC in simpler terms or the structure of a modern web-application
- PHP error reporting
- How to get a single player's rank based on the score
- Do you really need to check for both isset() and empty() at the same time?
- Operator precedence or how does 'or die()' work.
- Numerical strings comparison
- What's wrong with popular articles telling you that foo is faster than bar?
- Why should I use prepared statements if escaping is safe?
- Do you abuse the null coalescing operator (and isset/empty as well)?

https://phpdelusions.net/articles/paths