

Loads environment variables from `.env` to `getenv()`, `$_ENV` and `$_SERVER` automagically.

191 commits

8 branches

20 releases

38 contributors

BSD-3-Clause

Branch: master ▾











New pull request

Create new file

Upload files

Find file

Clone or download ▾

 vlucas committed 13 days ago Merge pull request #197 from sephedo/master ...	Latest commit 90b2f41 13 days ago
 src	Merge pull request #197 from sephedo/master 13 days ago
 tests	Merge pull request #197 from sephedo/master 13 days ago
 .gitattributes	Fix occurrences of CRLF line endings. 2 years ago
 .gitignore	Tweaked git files 2 years ago
 .travis.yml	Fix PHP 5.3 on Travis CI 3 months ago
 LICENSE.txt	Initial working & tested phpdotenv 5 years ago
 README.md	Merge pull request #197 from sephedo/master 13 days ago
 composer.json	Use PHPUnit\Framework\TestCase instead of PHPUnit_Framework_TestCase 3 months ago
 phpunit.xml.dist	Update phpunit config 2 years ago

## README.md

# PHP dotenv

Loads environment variables from `.env` to `getenv()`, `$_ENV` and `$_SERVER` automagically.

This is a PHP version of the original [Ruby dotenv](#).

build passing

## Why .env?

You should never store sensitive credentials in your code. Storing [configuration in the environment](#) is one of the tenets of a [twelve-factor app](#). Anything that is likely to change between deployment environments – such as database credentials or credentials for 3rd party services – should be extracted from the code into environment variables.

Basically, a `.env` file is an easy way to load custom configuration variables that your application needs without having to modify `.htaccess` files or Apache/nginx virtual hosts. This means you won't have to edit any files outside the project, and all the environment variables are always set no matter how you run your project - Apache, Nginx, CLI, and even PHP 5.4's built-in webserver. It's WAY easier than all the other ways you know of to set environment variables, and you're going to love it.

- NO editing virtual hosts in Apache or Nginx
- NO adding `php_value` flags to `.htaccess` files
- EASY portability and sharing of required ENV values
- COMPATIBLE with PHP's built-in web server and CLI runner

## Installation with Composer

```
curl -s http://getcomposer.org/installer | php
php composer.phar require vlucas/phpdotenv
```

## Usage

The `.env` file is generally kept out of version control since it can contain sensitive API keys and passwords. A separate `.env.example` file is created with all the required environment variables defined except for the sensitive ones, which are either user-supplied for their own development environments or are communicated elsewhere to project collaborators. The project collaborators then independently copy the `.env.example` file to a local `.env` and ensure all the settings are correct for their local environment, filling in the secret keys or providing their own values when necessary. In this usage, the `.env` file should be added to the project's `.gitignore` file so that it will never be committed by collaborators. This usage ensures that no sensitive passwords or API keys will ever be in the version control history so there is less risk of a security breach, and production values will never have to be shared with all project collaborators.

Add your application configuration to a `.env` file in the root of your project. **Make sure the `.env` file is added to your `.gitignore` so it is not checked-in the code**

```
S3_BUCKET="dotenv"
SECRET_KEY="souper_seekret_key"
```

Now create a file named `.env.example` and check this into the project. This should have the ENV variables you need to have set, but the values should either be blank or filled with dummy data. The idea is to let people know what variables are required, but not give them the sensitive production values.

```
S3_BUCKET="devbucket"
SECRET_KEY="abc123"
```

You can then load `.env` in your application with:

```
$dotenv = new Dotenv\Dotenv(__DIR__);
$dotenv->load();
```

Optionally you can pass in a filename as the second parameter, if you would like to use something other than `.env`

```
$dotenv = new Dotenv\Dotenv(__DIR__, 'myconfig');
$dotenv->load();
```

All of the defined variables are now accessible with the `getenv` method, and are available in the `$_ENV` and `$_SERVER` super-globals.

```
$s3_bucket = getenv('S3_BUCKET');
$s3_bucket = $_ENV['S3_BUCKET'];
$s3_bucket = $_SERVER['S3_BUCKET'];
```

You should also be able to access them using your framework's Request class (if you are using a framework).

```
$s3_bucket = $request->env('S3_BUCKET');
$s3_bucket = $request->getEnv('S3_BUCKET');
$s3_bucket = $request->server->get('S3_BUCKET');
$s3_bucket = env('S3_BUCKET');
```

## Nesting Variables

It's possible to nest an environment variable within another, useful to cut down on repetition.

This is done by wrapping an existing environment variable in `${...}` e.g.

```
BASE_DIR="/var/webroot/project-root"
CACHE_DIR="${BASE_DIR}/cache"
TMP_DIR="${BASE_DIR}/tmp"
```

## Immutability

By default, Dotenv will NOT overwrite existing environment variables that are already set in the environment.

If you want Dotenv to overwrite existing environment variables, use `overload` instead of `load` :

```
$dotenv = new Dotenv\Dotenv(__DIR__);
$dotenv->overload();
```

## Requiring Variables to be Set

Using Dotenv, you can require specific ENV vars to be defined (`$_ENV`, `$_SERVER` or `getenv()`) - throws an exception otherwise. Note: It does not check for existence of a variable in a '.env' file. This is particularly useful to let people know any explicit required variables that your app will not work without.

You can use a single string:

```
$dotenv->required('DATABASE_DSN');
```

Or an array of strings:

```
$dotenv->required(['DB_HOST', 'DB_NAME', 'DB_USER', 'DB_PASS']);
```

If any ENV vars are missing, Dotenv will throw a `RuntimeException` like this:

```
One or more environment variables failed assertions: DATABASE_DSN is missing
```

## Empty Variables

Beyond simply requiring a variable to be set, you might also need to ensure the variable is not empty:

```
$dotenv->required('DATABASE_DSN')->notEmpty();
```

If the environment variable is empty, you'd get an Exception:

```
One or more environment variables failed assertions: DATABASE_DSN is empty
```

## Integer Variables

You might also need to ensure that the variable is of an integer value. You may do the following:

```
$dotenv->required('FOO')->isInteger();
```

If the environment variable is not an integer, you'd get an Exception:

```
One or more environment variables failed assertions: FOO is not an integer
```

## Boolean Variables

You may need to ensure a variable is in the form of a boolean, accepting "On", "1", "Yes", "Off", "0" and "No". You may do the following:

```
$dotenv->required('FOO')->isBoolean();
```

If the environment variable is not a boolean, you'd get an Exception:

```
One or more environment variables failed assertions: FOO is not a boolean
```

## Allowed Values

It is also possible to define a set of values that your environment variable should be. This is especially useful in situations where only a handful of options or drivers are actually supported by your code:

```
$dotenv->required('SESSION_STORE')->allowedValues(['Filesystem', 'Memcached']);
```

If the environment variable wasn't in this list of allowed values, you'd get a similar Exception:

```
One or more environment variables failed assertions: SESSION_STORE is not an
allowed value
```

## Comments

You can comment your `.env` file using the `#` character. E.g.

```
# this is a comment
VAR="value" # comment
VAR=value # comment
```

## Usage Notes

When a new developer clones your codebase, they will have an additional **one-time step** to manually copy the `.env.example` file to `.env` and fill-in their own values (or get any sensitive values from a project co-worker).

phpdotenv is made for development environments, and generally should not be used in production. In production, the actual environment variables should be set so that there is no overhead of loading the `.env` file on each request. This can be achieved via an automated deployment process with tools like Vagrant, chef, or Puppet, or can be set manually with cloud hosts like Pagodabox and Heroku.

## Command Line Scripts

If you need to use environment variables that you have set in your `.env` file in a command line script that doesn't use the Dotenv library, you can `source` it into your local shell session:

```
source .env
```

## Contributing

1. Fork it
2. Create your feature branch ( `git checkout -b my-new-feature` )
3. Make your changes
4. Run the tests, adding new ones for your own code if necessary ( `phpunit` )
5. Commit your changes ( `git commit -am 'Added some feature'` )
6. Push to the branch ( `git push origin my-new-feature` )
7. Create new Pull Request