PHP: The Right Way

Welcome

There's a lot of outdated information on the Web that leads new PHP users astray, propagating bad practices and bad code. This must stop. *PHP: The Right Way* is an easy-to-read, quick reference for PHP best practices, accepted coding standards, and links to authoritative tutorials around the Web.

Getting Started

Use the Current Stable Version (5.4)

If you are just getting started with PHP make sure to start with the current stable release of PHP 5.4. PHP has made great strides adding powerful new features over the last few years. Don't let the minor version number difference between 5.2 and 5.4 fool you, it represents *major* improvements. If you are looking for a function or it's usage, the documentation on the php.net website will have the answer.

Built-in web server

You can start learning PHP without the hassle of installing and configuring a full-fledged web server (PHP 5.4 required). To start the server, run the following from your terminal in your project's web root:

```
> php -S localhost:8000
```

- Learn about the built-in, command line web server

Mac Setup

OSX comes prepackaged with PHP but it is normally a little behind the latest stable. Lion comes with PHP 5.3.6 and Mountain Lion has 5.3.10.

To update PHP on OSX you can get it installed through a number of Mac package managers, with php-osx by Liip being recommended.

The other option is to compile it yourself, in that case be sure to have installed either Xcode or Apple's substitute "Command Line Tools for Xcode" downloadable from Apple's Mac Developer Center.

For a complete "all-in-one" package including PHP, Apache web server and MySQL database, all this with a nice control GUI, try MAMP.

Windows Setup

PHP is available in several ways for Windows. You can download the binaries and until recently you could use a '.msi' installer. The installer is no longer supported and stops at PHP 5.3.0.

For learning and local development you can use the built in webserver with PHP 5.4 so you don't need to worry about configuring it. If you would like an "all-in-one" which includes a full-blown webserver and MySQL too then tools such as the Web Platform Installer, Zend Server CE, XAMPP and WAMP will help get a Windows development environment up and running fast. That said, these tools will be a little different from production so be careful of environment differences if you are working on Windows and deploying to Linux.

If you need to run your production system on Windows then IIS7 will give you the most stable and best performance. You can use phpmanager (a GUI plugin for IIS7) to make configuring and managing PHP simple. IIS7 comes with FastCGI built in and ready to go, you just need to configure PHP as a handler. For support and additional resources there is a dedicated area on iis.net for PHP.

Vagrant

Running your application on different environments in development and production can lead to strange bugs popping up when you go live. It's also tricky to keep different development environments up to date with the same version for all libraries used when working with a team of developers.

If you are developing on Windows and deploying to Linux (or anything non-Windows) or are developing in a team, you should consider using a virtual machine. This sounds tricky, but using Vagrant you can set up a simple virtual machine with only a few steps. These base boxes can then be set up manually, or you can use "provisioning" software such as Puppet or Chef to do this for you. Provisioning the base box is a great way to ensure that multiple boxes are set up in an identical fashion and removes the need for you to maintain complicated "set up" command lists. You can also "destroy" your base box and recreate it without many manual steps, making it easy to create a "fresh" installation.

Vagrant creates shared folders used to share your code between your host and your virtual machine, meaning you can create and edit your files on your host machine and then run the code inside your virtual machine.

Back to Top

Code Style Guide

The PHP community is large and diverse, composed of innumerable libraries, frameworks, and components. It is common for PHP developers to choose several of these and combine them into a single project. It is important that PHP code adhere (as close as possible) to a common code style to make it easy for developers to mix and match various libraries for their projects.

The Framework Interop Group has proposed and approved a series of style recommendations, known as PSR-0, PSR-1 and PSR-2. Don't let the funny names confuse you, these recommendations are merely a set of rules that some projects like Drupal, Zend, Symfony, CakePHP, phpBB, AWS SDK, FuelPHP, Lithium, etc are starting to adopt. You can use them for your own projects, or continue to use your own personal style.

Ideally you should write PHP code that adheres to a known standard. This could be any combination of PSR's, or one of the coding standards made by PEAR or Zend. This means other developers can easily read and work with your code, and applications that implement the components can have consistency even when working with lots of third-party code.

- Read about PSR-0
- Read about PSR-1
- Read about PSR-2
- Read about PEAR Coding Standards
- Read about Zend Coding Standards

You can use PHP_CodeSniffer to check code against any one of these recommendations, and plugins for text editors like Sublime Text 2 to be given real time feedback.

Use Fabien Potencier's PHP Coding Standards Fixer to automatically modify your code syntax so that it conforms with these standards, saving you from fixing each problem by hand.

English is preferred for all symbol names and code infrastructure. Comments may be written in any language easily readable by all current and future parties who may be working on the codebase.

Back to Top

Language Highlights

Programming Paradigms

PHP is a flexible, dynamic language that supports a variety of programming techniques. It has evolved dramatically over the years, notably adding a solid object-oriented model in PHP 5.0 (2004), anonymous functions and namespaces in PHP 5.3 (2009), and traits in PHP 5.4 (2012).

Object-oriented Programming

PHP has a very complete set of object-oriented programming features including support for classes, abstract classes, interfaces, inheritance, constructors, cloning, exceptions, and more.

- Read about Object-oriented PHP
- Read about Traits

Functional Programming

PHP supports first-class function, meaning that a function can be assigned to a variable. Both user defined and built-in functions can be referenced by a variable and invoked dynamically. Functions can be passed as arguments to other functions (feature called Higher-order functions) and function can return other functions.

Recursion, a feature that allows a function to call itself is supported by the language, but most of the PHP code focus on iteration.

New anonymous functions (with support for closures) are present since PHP 5.3 (2009).

PHP 5.4 added the ability to bind closures to an object's scope and also improved support for callables such that they can be used interchangeably with anonymous functions in almost all cases.

- Continue reading on Functional Programming in PHP
- Read about Anonymous Functions
- Read about the Closure class
- More details in the Closures RFC
- Read about Callables
- Read about dynamically invoking functions with `call_user_func_array`

Meta Programming

PHP supports various forms of meta programming through mechanisms like the Reflection API and Magic Methods. There are many Magic Methods available like `__get()`, `__set()`, `__clone()`, `__toString()`, `__invoke()`, etc. that allow developers to hook into class behavior. Ruby developers often say that PHP is lacking `method_missing`, but it is available as `__call()` and `__callStatic()`.

- Read about Magic Methods
- Read about Reflection

Namespaces

As mentioned above, the PHP community has a lot of developers creating lots of code. This means that one library's PHP code may use the same class name as another library. When both libraries are used in the same namespace, they collide and cause trouble.

Namespaces solve this problem. As described in the PHP reference manual, namespaces may be compared to operating system directories that *namespace* files; two files with the same name may co-exist in separate directories. Likewise, two PHP classes with the same name may co-exist in separate PHP namespaces. It's as simple as that.

It is important for you to namespace your code so that it may be used by other developers without fear of colliding with other libraries.

One recommended way to use namespaces is outlined in PSR-0, which aims to provide a standard file, class and namespace convention to allow plug-and-play code.

- Read about Namespaces
- Read about PSR-0

Standard PHP Library

The Standard PHP Library (SPL) is packaged with PHP and provides a collection of classes and interfaces. It is made up primarily of commonly needed

datastructure classes (stack, queue, heap, and so on), and iterators which can traverse over these datastructures or your own classes which implement SPL interfaces.

- Read about the SPL

Command Line Interface

PHP was created primarily to write web applications, but it's also useful for scripting command line interface (CLI) programs. Command line PHP programs can help you automate common tasks like testing, deployment, and application administrativia.

CLI PHP programs are powerful because you can use your app's code directly without having to create and secure a web GUI for it. Just be sure not to put your CLI PHP scripts in your public web root!

Try running PHP from your command line:

```
> php -i
```

The `-i` option will print your PHP configuration just like the `phpinfo` function.

The `-a` option provides an interactive shell, similar to ruby's IRB or python's interactive shell. There are a number of other useful command line options, too.

Let's write a simple "Hello, $name" CLI program. To try it out, create a file named `hello.php`, as below.

```
<?php
if ($argc != 2) {
    echo "Usage: php hello.php [name].\n";
    exit(1);
}
$name = $argv[1];
echo "Hello, $name\n";
```

PHP sets up two special variables based on the arguments your script is run with. `$argc` is an integer variable containing the argument *count* and `$argv` is an array variable containing each argument's *value*. The first argument is always the name of your PHP script file, in this case `hello.php`.

The `exit()` expression is used with a non zero number to let the shell know that the command failed. Commonly used exit codes can be found here

To run our script, above, from the command line:

```
> php hello.php
Usage: php hello.php [name]
> php hello.php world
Hello, world
```

- Learn about running PHP from the command line
- Learn about setting up Windows to run PHP from the command line

XDebug

One of the most useful tools in software development is a proper debugger. It allows you to trace the execution of your code and monitor the contents of the stack. XDebug, PHP's debugger, can be utilized by various IDEs to provide Breakpoints and stack inspection. It can also allow tools like PHPUnit and KCacheGrind to perform code coverage analysis and code profiling.

If you find yourself in a bind, willing to resort to var_dump/print_r, and you still can't find the solution - maybe you need to use the debugger.

Installing XDebug can be tricky, but one of its most important features is "Remote Debugging" - if you develop code locally and then test it inside a VM or on another server, Remote Debugging is the feature that you will want to enable right away.

Traditionally, you will modify your Apache VHost or .htaccess file with these values:

```
php_value xdebug.remote_host=192.168.?.?
php_value xdebug.remote_port=9000
```

The "remote host" and "remote port" will correspond to your local computer and the port that you configure your IDE to listen on. Then it's just a matter of putting your IDE into "listen for connections" mode, and loading the URL:

```
http://your-website.example.com/index.php?XDEBUG_SESSION_START=1
```

Your IDE will now intercept the current state as the script executes, allowing you to set breakpoints and probe the values in memory.

- Learn more about XDebug

Back to Top

Dependency Management

There are a ton of PHP libraries, frameworks, and components to choose from. Your project will likely use several of them — these are project dependencies. Until recently, PHP did not have a good way to manage these project dependencies. Even if you managed them manually, you still had to worry about autoloaders. No more.

Currently there are two major package management systems for PHP - Composer and PEAR. Which one is right for you? The answer is both.

- Use **Composer** when managing dependencies for a single project.
- Use **PEAR** when managing dependencies for PHP as a whole on your system.

In general, Composer packages will be available only in the projects that you explicitly specify whereas a PEAR package would be available to all of your PHP projects. While PEAR might sound like the easier approach at first glance, there are advantages to using a project-by-project approach to your dependencies.

Composer and Packagist

Composer is a **brilliant** dependency manager for PHP. List your project's dependencies in a `composer.json` file and, with a few simple commands, Composer will automatically download your project's dependencies and setup autoloading for you.

There are already a lot of PHP libraries that are compatible with Composer, ready to be used in your project. These "packages" are listed on Packagist, the official repository for Composer-compatible PHP libraries.

How to Install Composer

You can install Composer locally (in your current working directory; though this is no longer recommended) or globally (e.g. /usr/local/bin). Let's assume you want to install Composer locally. From your project's root directory:

```
curl -s https://getcomposer.org/installer | php
```

This will download `composer.phar` (a PHP binary archive). You can run this with `php` to manage your project dependencies. **Please Note:** If you pipe downloaded code directly into an interpreter, please read the code online first to confirm it is safe.

How to Install Composer (manually)

Manually installing Composer is an advanced technique; however, there are various reasons why a developer might prefer this method vs. using the interactive installation routine. The interactive installation checks your PHP installation to ensure that:

- a sufficient version of PHP is being used
- `.phar` files can be executed correctly
- certain directory permissions are sufficient
- certain problematic extensions are not loaded
- certain `php.ini` settings are set

Since a manual installation performs none of these checks, you have to decide whether the trade-off is worth it for you. As such, below is how to obtain Composer manually:

```
curl -s https://getcomposer.org/composer.phar -o $HOME/local/bin/composer
chmod +x $HOME/local/bin/composer
```

The path `$HOME/local/bin` (or a directory of your choice) should be in your `$PATH` environment variable. This will result in a `composer` command being available.

When you come across documentation that states to run Composer as `php composer.phar install`, you can substitute that with:

```
composer install
```

How to Define and Install Dependencies

Composer keeps track of your project's dependencies in a file called `composer.json`. You can manage it by hand if you like, or use Composer itself. The `php composer.phar require` command adds a project dependency and if you don't have a `composer.json` file, one will be created. Here's an example that adds Twig as a dependency of your project. Run it in your project's root directory where you've downloaded `composer.phar`:

```
php composer.phar require twig/twig:~1.8
```

Alternatively the `php composer.phar init` command will guide you through creating a full `composer.json` file for your project. Either way, once you've created your `composer.json` file you can tell Composer to download and install your dependencies into the `vendors/` directory. This also applies to projects you've downloaded that already provide a `composer.json` file:

```
php composer.phar install
```

Next, add this line to your application's primary PHP file; this will tell PHP to use Composer's autoloader for your project dependencies.

```
<?php
require 'vendor/autoload.php';
```

Now you can use your project dependencies, and they'll be autoloaded on demand.

Updating your dependencies

Composer creates a file called `composer.lock` which stores the exact version of each package it downloaded when you first ran `php composer.phar install`. If you share your project with other coders and the `composer.lock` file is part of your distribution, when they run `php composer.phar install` they'll get the same versions as you. To update your dependencies, run `php composer.phar update`.

This is most useful when you define your version requirements flexibly. For instance a version requirement of ~1.8 means "anything newer than 1.8.0, but less than 2.0.x-dev". You can also use the `*` wildcard as in `1.8.*`. Now Composer's `php composer.phar update` command will upgrade all your dependencies to the newest version that fits the restrictions you define.

Checking your dependencies for security issues

The Security Advisories Checker is a web service and a command-line tool, both will examine your `composer.lock` file and tell you if you need to update any of your dependencies.

- Learn about Composer

PEAR

Another veteran package manager that many PHP developers enjoy is PEAR. It behaves much the same way as Composer, but has some noteable differences.

PEAR requires each package to have a specific structure, which means that the author of the package must prepare it for usage with PEAR. Using a project which was not prepared to work with PEAR is not possible.

PEAR installs packages globally, which means after installing them once they are available to all projects on that server. This can be good if many projects rely on the same package with the same version but might lead to problems if version conflicts between two projects arise.

How to install PEAR

You can install PEAR by downloading the phar installer and executing it. The PEAR documentation has detailed install instructions for every operating system.

If you are using Linux, you can also have a look at your distribution package manager. Debian and Ubuntu for example have a apt `php-pear` package.

How to install a package

If the package is listed on the PEAR packages list, you can install it by specifying the official name:

```
pear install foo
```

If the package is hosted on another channel, you need to `discover` the channel first and also specify it when installing. See the Using channel docs for more information on this topic.

- Learn about PEAR

Handling PEAR dependencies with Composer

If you are already using Composer and you would like to install some PEAR code too, you can use Composer to handle your PEAR dependencies. This example will install code from `pear2.php.net`:

```
{
    "repositories": [
        {
            "type": "pear",
            "url": "http://pear2.php.net"
        }
    ],
    "require": {
        "pear-pear2/PEAR2_Text_Markdown": "*",
        "pear-pear2/PEAR2_HTTP_Request": "*"
    }
}
```

The first section `"repositories"` will be used to let Composer know it should "initialise" (or "discover" in PEAR terminology) the pear repo. Then the require section will prefix the package name like this:

    pear-channel/Package

The "pear" prefix is hardcoded to avoid any conflicts, as a pear channel could be the same as another packages vendor name for example, then the channel short name (or full URL) can be used to reference which channel the package is in.

When this code is installed it will be available in your vendor directory and automatically available through the Composer autoloader:

    vendor/pear-pear2.php.net/PEAR2_HTTP_Request/pear2/HTTP/Request.php

To use this PEAR package simply reference it like so:

```
$request = new pear2\HTTP\Request();
```

- Learn more about using PEAR with Composer

Back to Top

Coding Practices

The Basics

PHP is a vast language that allows coders of all levels the ability to produce code not only quickly, but efficiently. However while advancing through the language, we often forget the basics that we first learnt (or overlooked) in favor of short cuts and/or bad habits. To help combat this common issue, this section is aimed at reminding coders of the basic coding practices within PHP.

- Continue reading on The Basics

Date and Time

PHP has a class named DateTime to help you when reading, writing, comparing or calculating with date and time. There are many date and time related functions in PHP besides DateTime, but it provides nice object-oriented interface to most common uses. It can handle time zones, but that is outside this short introduction.

To start working with DateTime, convert raw date and time string to an object with `createFromFormat()` factory method or do `new \DateTime` to get the current date and time. Use `format()` method to convert DateTime back to a string for output.

```
<?php
$raw = '22. 11. 1968';
$start = \DateTime::createFromFormat('d. m. Y', $raw);

echo 'Start date: ' . $start->format('m/d/Y') . "\n";
```

Calculating with DateTime is possible with the DateInterval class. DateTime has methods like `add()` and `sub()` that take a DateInterval as an argument. Do not write code that expect same number of seconds in every day, both daylight saving and timezone alterations will break that assumption. Use date intervals instead. To calculate date difference use the `diff()` method. It will return new DateInterval, which is super easy to display.

```
<?php
// create a copy of $start and add one month and 6 days
$end = clone $start;
$end->add(new \DateInterval('P1M6D'));

$diff = $end->diff($start);
echo 'Difference: ' . $diff->format('%m month, %d days (total: %a days)') . "\n";
// Difference: 1 month, 6 days (total: 37 days)
```

On DateTime objects you can use standard comparison:

```
<?php
if ($start < $end) {
    echo "Start is before end!\n";
}
```

One last example to demonstrate the DatePeriod class. It is used to iterate over recurring events. It can take two DateTime objects, start and end, and the interval for which it will return all events in between.

```
<?php
// output all thursdays between $start and $end
$periodInterval = \DateInterval::createFromDateString('first thursday');
$periodIterator = new \DatePeriod($start, $periodInterval, $end, \DatePeriod::EXCLUDE_START_
foreach ($periodIterator as $date) {
    // output each date in the period
    echo $date->format('m/d/Y') . ' ';
}
```

- Read about DateTime
- Read about date formatting (accepted date format string options)

Design Patterns

When you are building your application it is helpful to use common patterns in your code and common patterns for the overall structure of your project. Using common patterns is helpful because it makes it much easier to manage your code and lets other developers quickly understand how everything fits together.

If you use a framework then most of the higher level code and project structure will be based on that framework, so a lot of the pattern decisions are made for you. But it is still up to you to pick out the best patterns to follow in the code you build on top of the framework. If, on the other hand, you are not using a framework to build your application then you have to find the patterns that best suit the type and size of application that you're building.

- Continue reading on Design Patterns

Exceptions

Exceptions are a standard part of most popular programming languages, but they are often overlooked by PHP programmers. Languages like Ruby are extremely Exception heavy, so whenever something goes wrong such as a HTTP request failing, or a DB query goes wrong, or even if an image asset could not be found, Ruby (or the gems being used) will throw an exception to the screen meaning you instantly know there is a mistake.

PHP itself is fairly lax with this, and a call to `file_get_contents()` will usually just get you a `FALSE` and a warning. Many older PHP frameworks like CodeIgniter will just return a false, log a message to their proprietary logs and maybe let you use a method like `$this->upload->get_error()` to see what went wrong. The problem here is that you have to go looking for a mistake and check the docs to see what the error method is for this class, instead of having it made extremely obvious.

Another problem is when classes automatically throw an error to the screen and exit the process. When you do this you stop another developer from being able to dynamically handle that error. Exceptions should be thrown to make a developer aware of an error, then they can choose how to handle this. E.g:

```
<?php
$email = new Fuel\Email;
$email->subject('My Subject');
$email->body('How the heck are you?');
```

```
$email->to('guy@example.com', 'Some Guy');

try
{
    $email->send();
}
catch(Fuel\Email\ValidationFailedException $e)
{
    // The validation failed
}
catch(Fuel\Email\SendingFailedException $e)
{
    // The driver could not send the email
}
```

SPL Exceptions

The generic `Exception` class provides very little debugging context for the developer; however, to remedy this, it is possible to create a specialized `Exception` type by sub-classing the generic `Exception` class:

```
<?php
class ValidationException extends Exception {}
```

This means you can add multiple catch blocks and handle different Exceptions differently. This can lead to the creation of a *lot* of custom Exceptions, some of which could have been avoided using the SPL Exceptions provided in the SPL extension.

If for example you use the `__call()` Magic Method and an invalid method is requested then instead of throwing a standard Exception which is vague, or creating a custom Exception just for that, you could just `throw new BadFunctionCallException;`.

- Read about Exceptions
- Read about SPL Exceptions
- Nesting Exceptions In PHP
- Exception Best Practices in PHP 5.3

Back to Top

Databases

Many times your PHP code will use a database to persist information. You have a few options to connect and interact with your database. The recommended

option *until PHP 5.1.0* was to use native drivers such as mysql, mysqli, pgsql, etc.

Native drivers are great if you are only using ONE database in your application, but if, for example, you are using MySQL and a little bit of MSSQL, or you need to connect to an Oracle database, then you will not be able to use the same drivers. You'll need to learn a brand new API for each database — and that can get silly.

As an extra note on native drivers, the mysql extension for PHP is no longer in active development, and the official status since PHP 5.4.0 is "Long term deprecation". This means it will be removed within the next few releases, so by PHP 5.6 (or whatever comes after 5.5) it may well be gone. If you are using `mysql_connect()` and `mysql_query()` in your applications then you will be faced with a rewrite at some point down the line, so the best option is to replace mysql usage with mysqli or PDO in your applications within your own development schedules so you won't be rushed later on. *If you are starting from scratch then absolutely do not use the mysql extension: use the MySQLi extension, or use PDO.*

- PHP: Choosing an API for MySQL

PDO

PDO is a database connection abstraction library — built into PHP since 5.1.0 — that provides a common interface to talk with many different databases. PDO will not translate your SQL queries or emulate missing features; it is purely for connecting to multiple types of database with the same API.

More importantly, `PDO` allows you to safely inject foreign input (e.g. IDs) into your SQL queries without worrying about database SQL injection attacks. This is possible using PDO statements and bound parameters.

Let's assume a PHP script receives a numeric ID as a query parameter. This ID should be used to fetch a user record from a database. This is the `wrong` way to do this:

```
<?php
$pdo = new PDO('sqlite:users.db');
$pdo->query("SELECT name FROM users WHERE id = " . $_GET['id']); // <-- NO!
```

This is terrible code. You are inserting a raw query parameter into a SQL query. This will get you hacked in a heartbeat. Just imagine if a hacker passes in an inventive `id` parameter by calling a URL like `http://domain.com/?id=1%3BDELETE+FROM+users`. This will set the `$_GET['id']` variable to `1;DELETE FROM users` which will delete all of your users! Instead, you should sanitize the ID input using PDO bound parameters.

```php
<?php
$pdo = new PDO('sqlite:users.db');
$stmt = $pdo->prepare('SELECT name FROM users WHERE id = :id');
$stmt->bindParam(':id', $_GET['id'], PDO::PARAM_INT); //<-- Automatically sanitized by PDO
$stmt->execute();
```

This is correct code. It uses a bound parameter on a PDO statement. This escapes the foreign input ID before it is introduced to the database preventing potential SQL injection attacks.

- Learn about PDO

You should also be aware that database connections use up resources and it was not unheard-of to have resources exhausted if connections were not implicitly closed, however this was more common in other languages. Using PDO you can implicitly close the connection by destroying the object by ensuring all remaining references to it are deleted, ie. set to NULL. If you don't do this explicitly, PHP will automatically close the connection when your script ends unless of course you are using persistent connections.

- Learn about PDO connections

Abstraction Layers

Many frameworks provide their own abstraction layer which may or may not sit on top of PDO. These will often emulate features for one database system that another is missing from another by wrapping your queries in PHP methods, giving you actual database abstraction. This will of course add a little overhead, but if you are building a portable application that needs to work with MySQL, PostgreSQL and SQLite then a little overhead will be worth it the sake of code cleanliness.

Some abstraction layers have been built using the PSR-0 namespace standard so can be installed in any application you like:

- Aura SQL
- Doctrine2 DBAL
- ZF2 Db
- ZF1 Db

Back to Top

Security

Web Application Security

There are bad people ready and willing to exploit your web application. It is important that you take necessary precautions to harden your web application's security. Luckily, the fine folks at The Open Web Application Security Project (OWASP) have compiled a comprehensive list of known security issues and methods to protect yourself against them. This is a must read for the security-conscious developer.

- Read the OWASP Security Guide

Password Hashing

Eventually everyone builds a PHP application that relies on user login. Usernames and passwords are stored in a database and later used to authenticate users upon login.

It is important that you properly *hash* passwords before storing them. Password hashing is an irreversible, one way function performed against the users password. This produces a fixed-length string that can not be feasibly reversed. This means you can compare a hash against another to determine if they both came from the same source string, but you can not determine the original string. If passwords are not hashed and your database is accessed by an unauthorized third-party, all user accounts are now compromised. Some users may (unfortunately) use the same password for other services. Therefore, it is important to take security seriously.

Hashing passwords with `password_hash`

In PHP 5.5 `password_hash` will be introduced. At this time it is using BCrypt, the strongest algorithm currently supported by PHP. It will be updated in the future to support more algorithms as needed though. The `password_compat` library was created to provide forward compatibility for PHP >= 5.3.7.

Below we hash a string, we then check the hash against a new string. Because our two source strings are different ('secret-password' vs. 'bad-password') this login will fail.

```
<?php
require 'password.php';

$passwordHash = password_hash('secret-password', PASSWORD_DEFAULT);
```

```
if (password_verify('bad-password', $passwordHash)) {
    //Correct Password
} else {
    //Wrong password
}
```

- Learn about `password_hash`
- `password_compat` for PHP >= 5.3.7 && < 5.5
- Learn about hashing in regards to cryptography
- PHP `password_hash` RFC

Data Filtering

Never ever (ever) trust foreign input introduced to your PHP code. Always sanitize and validate foreign input before using it in code. The `filter_var` and `filter_input` functions can sanitize text and validate text formats (e.g. email addresses).

Foreign input can be anything: `$_GET` and `$_POST` form input data, some values in the `$_SERVER` superglobal, and the HTTP request body via `fopen('php://input', 'r')`. Remember, foreign input is not limited to form data submitted by the user. Uploaded and downloaded files, session values, cookie data, and data from third-party web services are foreign input, too.

While foreign data can be stored, combined, and accessed later, it is still foreign input. Every time you process, output, concatenate, or include data in your code, ask yourself if the data is filtered properly and can it be trusted.

Data may be *filtered* differently based on its purpose. For example, when unfiltered foreign input is passed into HTML page output, it can execute HTML and JavaScript on your site! This is known as Cross-Site Scripting (XSS) and can be a very dangerous attack. One way to avoid XSS is to sanitize all user-generated data before outputting it to your page by removing HTML tags with the `strip_tags` function or escaping characters with special meaning into their respective HTML entities with the `htmlentities` or `htmlspecialchars` functions.

Another example is passing options to be executed on the command line. This can be extremely dangerous (and is usually a bad idea), but you can use the built-in `escapeshellarg` function to sanitize the executed command's arguments.

One last example is accepting foreign input to determine a file to load from the filesystem. This can be exploited by changing the filename to a file path. You need to remove "/", "../", null bytes, or other characters from the file path so it can't load hidden, non-public, or sensitive files.

- Learn about data filtering

- Learn about `filter_var`
- Learn about `filter_input`
- Learn about handling null bytes

Sanitization

Sanitization removes (or escapes) illegal or unsafe characters from foreign input.

For example, you should sanitize foreign input before including the input in HTML or inserting it into a raw SQL query. When you use bound parameters with PDO, it will sanitize the input for you.

Sometimes it is required to allow some safe HTML tags in the input when including it in the HTML page. This is very hard to do and many avoid it by using other more restricted formatting like Markdown or BBCode, although whitelisting libraries like HTML Purifier exists for this reason.

See Sanitization Filters

Validation

Validation ensures that foreign input is what you expect. For example, you may want to validate an email address, a phone number, or age when processing a registration submission.

See Validation Filters

Configuration Files

When creating configuration files for your applications, best practices recommend that one of the following methods be followed:

- It is recommended that you store your configuration information where it cannot be accessed directly and pulled in via the file system.
- If you must store your configuration files in the document root, name the files with a `.php` extension. This ensures that, even if the script is accessed directly, it will not be outputed as plain text.
- Information in configuration files should be protected accordingly, either through encryption or group/user file system permissions

Register Globals

NOTE: As of PHP 5.4.0 the `register_globals` setting has been removed and can no longer be used. This is only included as a warning for anyone in the process of upgrading a legacy application.

When enabled, the `register_globals` configuration setting that makes several types of variables (including ones from `$_POST`, `$_GET` and `$_REQUEST`) available in the global scope of your application. This can easily lead to security issues as your application cannot effectively tell where the data is coming from.

For example: `$_GET['foo']` would be available via `$foo`, which can override variables that have not been declared. If you are using PHP $< 5.4.0$ **make sure** that `register_globals` is **off**.

- Register_globals in the PHP manual

Error Reporting

Error logging can be useful in finding the problem spots in your application, but it can also expose information about the structure of your application to the outside world. To effectively protect your application from issues that could be caused by the output of these messages, you need to configure your server differently in development versus production (live).

Development

To show every possible error during **development**, configure the following settings in your `php.ini`:

```
display_errors = On
display_startup_errors = On
error_reporting = -1
log_errors = On
```

> Passing in the value `-1` will show every possible error, even when new levels and constants are added in future PHP versions. The `E_ALL` constant also behaves this way as of PHP 5.4. - php.net

The `E_STRICT` error level constant was introduced in 5.3.0 and is not part of `E_ALL`, however it became part of `E_ALL` in 5.4.0. What does this mean? In terms of reporting every possible error in version 5.3 it means you must use either `-1` or `E_ALL | E_STRICT`.

Reporting every possible error by PHP version

- $<$ 5.3 `-1` or `E_ALL`
- 5.3 `-1` or `E_ALL | E_STRICT`
- $>$ 5.3 `-1` or `E_ALL`

Production

To hide errors on your **production** environment, configure your `php.ini` as:

```
display_errors = Off
display_startup_errors = Off
error_reporting = E_ALL
log_errors = On
```

With these settings in production, errors will still be logged to the error logs for the web server, but will not be shown to the user. For more information on these settings, see the PHP manual:

- error_reporting
- display_errors
- display_startup_errors
- log_errors

Back to Top

Testing

Writing automated tests for your PHP code is considered a best practice and can lead to well-built applications. Automated tests are a great tool for making sure your application does not break when you are making changes or adding new functionality and should not be ignored.

There are several different types of testing tools (or frameworks) available for PHP, which use different approaches - all of which are trying to avoid manual testing and the need for large Quality Assurance teams, just to make sure recent changes didn't break existing functionality.

Test Driven Development

From Wikipedia:

> Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes a failing automated test case that defines a desired improvement or new function, then produces code to pass that test and finally refactors the new code to acceptable standards. Kent Beck, who is credited with having developed or 'rediscovered' the technique, stated in 2003 that TDD encourages simple designs and inspires confidence

There are several different types of testing that you can do for your application

Unit Testing

Unit Testing is a programming approach to ensure functions, classes and methods are working as expected, from the point you build them all the way through the development cycle. By checking values going in and out of various functions and methods, you can make sure the internal logic is working correctly. By using Dependency Injection and building "mock" classes and stubs you can verify that dependencies are correctly used for even better test coverage.

When you create a class or function you should create a unit test for each behavior it must have. At a very basic level you should make sure it errors if you send it bad arguments and make sure it works if you send it valid arguments. This will help ensure that when you make changes to this class or function later on in the development cycle that the old functionality continues to work as expected. The only alternative to this would be var_dump() in a test.php, which is no way to build an application - large or small.

The other use for unit tests is contributing to open source. If you can write a test that shows broken functionality (i.e. fails), then fix it, and show the test passing, patches are much more likely to be accepted. If you run a project which accepts pull requests then you should suggest this as a requirement.

PHPUnit is the de-facto testing framework for writing unit tests for PHP applications, but there are several alternatives

- SimpleTest
- Enhance PHP
- PUnit
- atoum

Integration Testing

From Wikipedia:

> Integration testing (sometimes called Integration and Testing, abbreviated "I&T") is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

Many of the same tools that can be used for unit testing can be used for integration testing as many of the same principles are used.

Functional Testing

Sometimes also known as acceptance testing, functional testing consists of using tools to create automated tests that actually use your application instead of just verifying that individual units of code are behaving correctly and that individual units can speak to each other correctly. These tools typically work using real data and simulating actual users of the application.

Functional Testing Tools

- Selenium
- Mink
- Codeception is a full-stack testing framework that includes acceptance testing tools

Behavior Driven Development

There are two different types of Behavior-Driven Development (BDD): SpecBDD and StoryBDD. SpecBDD focuses on technical behavior or code, while StoryBDD focuses on business or feature behaviors or interactions. PHP has frameworks for both types of BDD.

With StoryBDD, you write human-readable stories that describe the behavior of your application. These stories can then be run as actual tests against your application. The framework used in PHP applications for StoryBDD is Behat, which is inspired by Ruby's Cucumber project and implements the Gherkin DSL for describing feature behavior.

With SpecBDD, you write specifications that describe how your actual code should behave. Instead of testing a function or method, you are describing how that function or method should behave. PHP offers the PHPSpec framework for this purpose. This framework is inspired by the RSpec project for Ruby.

BDD Links

- Behat, the StoryBDD framework for PHP, inspired by Ruby's Cucumber project;
- PHPSpec, the SpecBDD framework for PHP, inspired by Ruby's RSpec project;
- Codeception is a full-stack testing framework that uses BDD principles.

Complementary Testing Tools

Besides individual testing and behavior driven frameworks, there are also a number of generic frameworks and helper libraries useful for any preferred approach taken.

Tool Links

- Selenium is a browser automation tool which can be integrated with PHPUnit
- Mockery is a Mock Object Framework which can be integrated with PHPUnit or PHPSpec
- Prophecy is a highly opinionated yet very powerful and flexible PHP object mocking framework. It's integrated with PHPSpec and can be used with PHPUnit.

Back to Top

Servers and Deployment

PHP applications can be deployed and run on production web servers in a number of ways.

Platform as a Service (PaaS)

PaaS provides the system and network architecture necessary to run PHP applications on the web. This means little to no configuration for launching PHP applications or PHP frameworks.

Recently PaaS has become a popular method for deploying, hosting, and scaling PHP applications of all sizes. You can find a list of PHP PaaS "Platform as a Service" providers in our resources section.

Virtual or Dedicated Servers

If you are comfortable with systems administration, or are interested in learning it, virtual or dedicated servers give you complete control of your application's production environment.

nginx and PHP-FPM

PHP, via PHP's built-in FastCGI Process Manager (FPM), pairs really nicely with nginx, which is a lightweight, high-performance web server. It uses less memory than Apache and can better handle more concurrent requests. This is especially important on virtual servers that don't have much memory to spare.

- Read more on nginx
- Read more on PHP-FPM
- Read more on setting up nginx and PHP-FPM securely

Apache and PHP

PHP and Apache have a long history together. Apache is wildly configurable and has many available modules to extend functionality. It is a popular choice for shared servers and an easy setup for PHP frameworks and open source apps like WordPress. Unfortunately, Apache uses more resources than nginx by default and cannot handle as many visitors at the same time.

Apache has several possible configurations for running PHP. The most common and easiest to setup is the prefork MPM with mod_php5. While it isn't the most memory efficient, it is the simplest to get working and to use. This is probably the best choice if you don't want to dig too deeply into the server administration aspects. Note that if you use mod_php5 you MUST use the prefork MPM.

Alternatively, if you want to squeeze more performance and stability out of Apache then you can take advantage of the same FPM system as nginx and run the worker MPM or event MPM with mod_fastcgi or mod_fcgid. This configuration will be significantly more memory efficient and much faster but it is more work to set up.

- Read more on Apache
- Read more on Multi-Processing Modules
- Read more on mod_fastcgi
- Read more on mod_fcgid

Shared Servers

PHP has shared servers to thank for its popularity. It is hard to find a host without PHP installed, but be sure it's the latest version. Shared servers allow you and other developers to deploy websites to a single machine. The upside to this is that it has become a cheap commodity. The downside is that you never know what kind of a ruckus your neighboring tenants are going to create; loading down the server or opening up security holes are the main concerns. If your project's budget can afford to avoid shared servers you should.

Building and Deploying your Application

If you find yourself doing manual database schema changes or running your tests manually before updating your files (manually), think twice! With every additional manual task needed to deploy a new version of your app, the chances for potentially fatal mistakes increase. Whether you're dealing with a simple update, a comprehensive build process or even a continuous integration strategy, build automation is your friend.

Among the tasks you might want to automate are:

- Dependency management
- Compilation, minification of your assets
- Running tests
- Creation of documentation
- Packaging
- Deployment

Build Automation Tools

Build tools can be described as a collection of scripts that handle common tasks of software deployment. The build tool is not a part of your software, it acts on your software from 'outside'.

There are many open source tools available to help you with build automation, some are written in PHP others aren't. This shouldn't hold you back from using them, if they're better suited for the specific job. Here are a few examples:

Phing is the easiest way to get started with automated deployment in the PHP world. With Phing you can control your packaging, deployment or testing process from within a simple XML build file. Phing (which is based on Apache Ant) provides a rich set of tasks usually needed to install or update a web app and can be extended with additional custom tasks, written in PHP.

Capistrano is a system for *intermediate-to-advanced programmers* to execute commands in a structured, repeatable way on one or more remote machines. It is pre-configured for deploying Ruby on Rails applications, however people are **successfully deploying PHP systems** with it. Successful use of Capistrano depends on a working knowledge of Ruby and Rake.

Dave Gardner's blog post PHP Deployment with Capistrano is a good starting point for PHP developers interested in Capistrano.

Chef is more than a deployment framework, it is a very powerful Ruby based system integration framework that doesn't just deploy your app but can build your whole server environment or virtual boxes.

Chef resources for PHP developers:

- Three part blog series about deploying a LAMP application with Chef, Vagrant, and EC2
- Chef Cookbook which installs and configures PHP 5.3 and the PEAR package management system

Further reading:

- Automate your project with Apache Ant
- Maven, a build framework based on Ant and how to use it with PHP

Continuous Integration

> Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily — leading to multiple integrations per day. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

– *Martin Fowler*

There are different ways to implement continuous integration for PHP. Recently Travis CI has done a great job of making continuous integration a reality even for small projects. Travis CI is a hosted continuous integration service for the open source community. It is integrated with GitHub and offers first class support for many languages including PHP.

Further reading:

- Continuous Integration with Jenkins
- Continuous Integration with PHPCI
- Continuous Integration with Teamcity

Back to Top

Caching

PHP is pretty quick by itself, but bottlenecks can arise when you make remote connections, load files, etc. Thankfully, there are various tools available to speed up certain parts of your application, or reduce the number of times these various time consuming tasks need to run.

Bytecode Cache

When a PHP file is executed, under the hood it is first compiled to bytecode (also known as opcode) and, only then, the bytecode is executed. If a PHP file is not modified, the bytecode will always be the same. This means that the compilation step is a waste of CPU resources.

This is where Bytecode cache comes in. It prevents redundant compilation by storing bytecode in memory and reusing it on successive calls. Setting up bytecode cache is a matter of minutes, and your application will speed up significantly. There's really no reason not to use it.

Popular bytecodes caches are:

- APC
- XCache
- Zend Optimizer+ (part of Zend Server package)
- WinCache (extension for MS Windows Server)

Object Caching

There are times when it can be beneficial to cache individual objects in your code, such as with data that is expensive to get or database calls where the result is unlikely to change. You can use object caching software to hold these pieces of data in memory for extremely fast access later on. If you save these items to a data store after you retrieve them, then pull them directly from the cache for following requests, you can gain a significant improvement in performance as well as reduce the load on your database servers.

Many of the popular bytecode caching solutions let you cache custom data as well, so there's even more reason to take advantage of them. APC, XCache, and WinCache all provide APIs to save data from your PHP code to their memory cache.

The most commonly used memory object caching systems are APC and memcached. APC is an excellent choice for object caching, it includes a simple API for adding your own data to its memory cache and is very easy to setup and use. The one real limitation of APC is that it is tied to the server it's installed on. Memcached on the other hand is installed as a separate service and can be accessed across the network, meaning that you can store objects in a hyper-fast data store in a central location and many different systems can pull from it.

Note that when running PHP as a (Fast-)CGI application inside your webserver, every PHP process will have its own cache, i.e. APC data is not shared between your worker processes. In these cases, you might want to consider using memcached instead, as it's not tied to the PHP processes.

In a networked configuration APC will usually outperform memcached in terms of access speed, but memcached will be able to scale up faster and further. If you do not expect to have multiple servers running your application, or do not need the extra features that memcached offers then APC is probably your best choice for object caching.

Example logic using APC:

```
<?php
// check if there is data saved as 'expensive_data' in cache
$data = apc_fetch('expensive_data');
if ($data === false) {
    // data is not in cache; save result of expensive call for later use
    apc_add('expensive_data', $data = get_expensive_data());
}

print_r($data);
```

Learn more about popular object caching systems:

- APC Functions
- Memcached
- Redis
- XCache APIs
- WinCache Functions

Back to Top

Resources

From the Source

- PHP Website
- PHP Documentation

People to Follow

- Rasmus Lerdorf
- Fabien Potencier
- Derick Rethans
- Chris Shiflett
- Sebastian Bergmann

- Matthew Weier O'Phinney
- Pádraic Brady
- Anthony Ferrara
- Nikita Popov

Mentoring

- phpmentoring.org - Formal, peer to peer mentoring in the PHP community.

PHP PaaS Providers

- PagodaBox
- AppFog
- Heroku (PHP support is undocumented but based on stable Facebook partnership link)
- fortrabbit
- Engine Yard Orchestra PHP Platform
- Red Hat OpenShift Platform
- dotCloud
- AWS Elastic Beanstalk
- cloudControl
- Windows Azure
- Zend Developer Cloud
- Google App Engine

Frameworks

Rather than re-invent the wheel, many PHP developers use frameworks to build out web applications. Frameworks abstract away many of the low-level concerns and provide helpful, easy-to-use interfaces to complete common tasks.

You do not need to use a framework for every project. Sometimes plain PHP is the right way to go, but if you do need a framework then there are three main types available:

- Micro Frameworks
- Full-Stack Frameworks
- Component Frameworks

Micro-frameworks are essentially a wrapper to route a HTTP request to a callback, controller, method, etc as quickly as possible, and sometimes come

with a few extra libraries to assist development such as basic database wrappers and the like. They are prominently used to build remote HTTP services.

Many frameworks add a considerable number of features on top of what is available in a micro-framework and these are known Full-Stack Frameworks. These often come bundled with ORMs, Authentication packages, etc.

Component-based frameworks are collections of specialized and single-purpose libraries. Disparate component-based frameworks can be used together to make a micro- or full-stack framework.

- Popular PHP Frameworks

Components

As mentioned above "Components" are another approach to the common goal of creating, distributing and implementing shared code. Various component repositories exist, the main two of which are:

- Packagist
- PEAR

Both of these repositories have command line tools associated with them to help the installation and upgrade processes, and have been explained in more detail in the Dependency Management section.

There are also component-based frameworks, which allow you to use their components with minimal (or no) requirements. For example, you can use the FuelPHP Validation package, without needing to use the FuelPHP framework itself. These projects are essentially just another repository for reusable components:

- Aura
- FuelPHP (2.0 only)
- Laravel's "Illuminate Components"
- Symfony Components

Back to Top

- Josh Lockhart
- Kris Jordan
- Phil Sturgeon
- New Media Campaigns
-