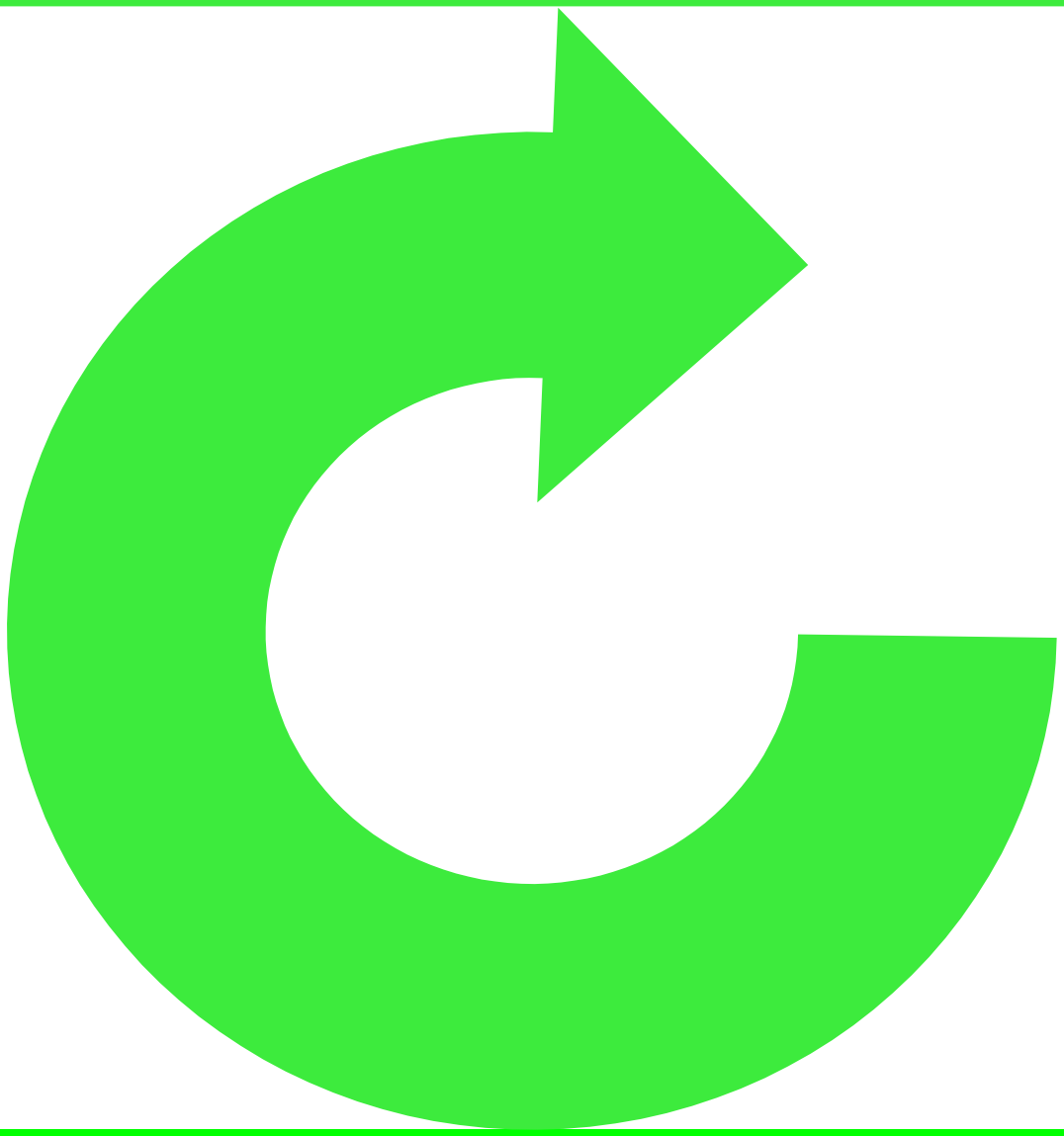
April 2014

PHP REBOOT



Storing Password Securely

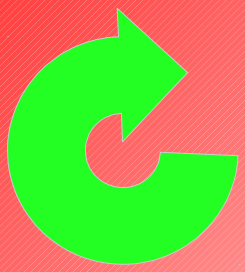
PHP Myths
Are they correct?

So you want to write test?

Apache Rewrite Module
Importance and basic use

Announcing first

PHP Reboot Meetup



Saturday
April 12th, 2014
at

EQUAL EXPERTS

Equal Experts India Pvt. Ltd.

Pune

Check http://www.meetup.com/PHPReboot for details
and RSVP for free

Meetup Topic:

Whole purpose of PHP Reboot is to learn/share new and advanced features of PHP and supporting technologies.

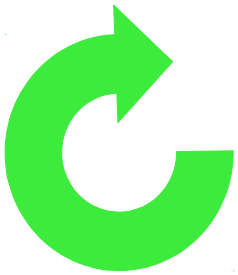
In this first session, we are trying to decide future path. So we will have following agenda for our first meetup:

1. PHP Myths (Discuss about issues with PHP; Real and advertised)
2. How latest changes in PHP overcome them?
3. What are best practices and why it is important to understand and follow them.
4. A little introduction of the topics to be discussed in the future.
5. What exactly local developers expect from local PHP community. This will be open session to decide the future of PHP Reboot community.

Contents

Fixed

Highlight



PHP Reboot

Editor:
Kapil sharma

Blogs republished (with permission):
Ben Dechrai, Chris Hartjes

Softwares used:
Libre Office Writer, Paint.net, Notepad++/Kate/vim on Windows 7 and Ubuntu 12.04.

Fonts used:
Mostly, Arial is used. However other fonts provided by Windows 7 or Libre Office Writer could be used at quite a few places. Copy of PHP

Downloading:
Copy of this file is available for free download on phpreboot.com.

Authors:
If you are experienced PHP developer and have some thing to share with community, we would like to request you to consider writing article/tutorial for PHPReboot. Just send a mail to phpreboot@gmail.com (email will change soon)

Website:
www.phpreboot.com

Notes:
We had taken maximum possible care to ensure all contents and code is perfect. How any author/publisher may not be held responsible for the correctness of contents and code.

Connect:
Twitter: @phpreboot
Facebook: facebook.com/PHReboot

Editorial

Hi,

Let me have an honor of announcing PHP Reboot magazine. Why another PHP magazine? Well PHP Reboot magazine is started as local PHP community magazine of PHP developers in Pune. Although anyone can download (yes its free) & read it, it is still a local Pune PHP community magazine.

So what does local mean? PHP is used globally. Here local just mean about local events, developers. PHP developers in Pune are initial contributors and readers and at least as of now, we are concentrating events in Pune.

It is not a professional magazine, designed and completely written in 'Libre Office Writer' so don't expect beautiful cover graphics or flashy material. Again, we are not going to register any copyright. All contents are copyright of original authors. Name of author with contact information is provided, in case you need further information of the license of articles.

PHP is almost two decades old language. With time, lot of tutorial, blogs were written about PHP. They were right at the time of writing but as PHP is changing rapidly, especially after PHP 5.3, many of old tutorial, blogs are now outdated. New developers are learning PHP from outdated stuff is a big concern. PHP reboot is an attempt to focus on new and latest features of PHP and provide latest information.

At the end, this is just a community effort. So we want to hear from the community. If you are having good PHP experience and can help community, please share your knowledge. You can write article about new and latest trends in PHP world as well as in related technologies like database (RDBMS, NoSQL, New SQL) or HTML/CSS or anything related to PHP. If you have some topic to share with community, don't hesitate, just send an email to phpreboot@gmail.com or share@phpreboot.com about your idea of article.

Kapil Sharma.

Under flashback, we list the popular blogs, news of last month. We have multiple source to find all good blogs and news but one of our major source is PHPDeveloper.org.

Below are the list of some of the latest happening in PHP world during March 2014.

1. **ClearCode: Symfony - Project Tamed**
   http://www.phpdeveloper.org/news/20963

   On the ClearCode blog today there's a new post for the Synfomy2 users out there with some recommendations about taming your project to make it more manageable and maintainable.

2. **ServerGrove Blog: Symfony2 components overview Finder**
   http://www.phpdeveloper.org/news/20960

   The ServerGrove blog has posted the latest in their series focusing in on the components in the Symfony2 framework. In this new post they look at the Finder component, used to locate files or directories in your project.

3. **Master Zend Framework: Simplifying Unit Testing (and asking for help when needed)**
   http://www.phpdeveloper.org/news/20933

   On *Matthew Ssetter*'s "Master Zend Framework" blog today he talks about simplifying unit testing and some of his experience with getting too complicated in his own testing practices.

4. **Hack: A new programming language for HHVM**
   https://code.facebook.com/posts/264544830379293/hack-a-new-programming-language-for-hhvm/

   In a hurry? Try Hack now: http://hacklang.org/

   Today we're releasing Hack, a programming language we developed for HHVM that interoperates seamlessly with PHP. Hack reconciles the fast development cycle of PHP with the discipline provided by static typing, while adding many features commonly found in other modern programming languages.

5. **SitePoint PHP Blog: CMS Showdown Nginx, Ghost, PHP and Phalcon**
   http://www.phpdeveloper.org/news/20934

   On the SitePoint PHP blog today *Bruno Skvorc* has written up the first part of his look at installing Ghost with Nginx and Phalcon on his hosting provider. This is the first post in his "showdown" series of trials on various CMS systems.

6. **Master Zend Framework: Simplifying Unit Testing (and asking for help when needed)**
   http://www.phpdeveloper.org/news/20933

   On *Matthew Ssetter*'s "Master Zend Framework" blog today he talks about simplifying unit testing and some of his experience with getting too complicated in his own testing practices.

7. **The PHP.cc Blog: Disintegration Testing**
   http://www.phpdeveloper.org/news/20932

   In this new post on thePHP.cc blog today *Sebastian Bergmann* relates the unfortunate disintegration of the Mars Climate Orbiter (back in 1999) back to a lesson on software testing and errors.

8. **Joshua Thijssen: Dynamic form modification in Symfony2**
   http://www.phpdeveloper.org/news/20931

   Joshua Thijssen has a new post to his site looking at a way to dynamically modify forms in a Symfony2-based application. Form handling can be a bit tricky (especially with more complex elements), and modifying them on the fly can be even more difficult.

9. **VG Tech: Comparing Your Privates in PHP**
   http://www.phpdeveloper.org/news/20925

   In a new post to their blog, the VG Tech folks talk about "comparing your privates" with a "hidden" feature of PHP. Don't worry, they're referring to private class properties on object instances here...

10. **Allan MacGregor: Exploring Traits**
    http://www.phpdeveloper.org/news/20917

    In his new post *Allan MacGregor* takes a look at a somewhat underused feature of PHP (since 5.4), traits. He talks about how they can help solve multiple inheritance issues and the power they can offer.

11. **Rob Allen: Use statements**
    http://www.phpdeveloper.org/news/20916

    Rob Allen's latest post focuses in on something that's been a part of PHP for a while now, back when namespacing was introduced - the "use" keyword. He shares some thoughts, both from others and himself, about whether or not they make code more readable.

12. **Liip Blog: Of HHVM, Hack and the future of PHP**
    http://www.phpdeveloper.org/news/20896

    Lukas Smith has posted some of his own thoughts on the Liip blog about the future of PHP, HHVM and Hack (related to this previous post from *Anthony Ferrara*) in the context of the company and the work they're doing.

13. **The PHP.cc Blog: PHPUnit 4.0 Test Proxies**
    http://www.phpdeveloper.org/news/20895

    On thePHP.cc blog today there's another post looking at an improvement in the latest release of the popular PHP unit testing tool, PHPUnit 4.0.0. In the post *Sebastian Bergmann* looks at test proxies.

14. **SitePoint PHP Blog: Risks and Challenges of Password Hashing**
    http://www.phpdeveloper.org/news/20889

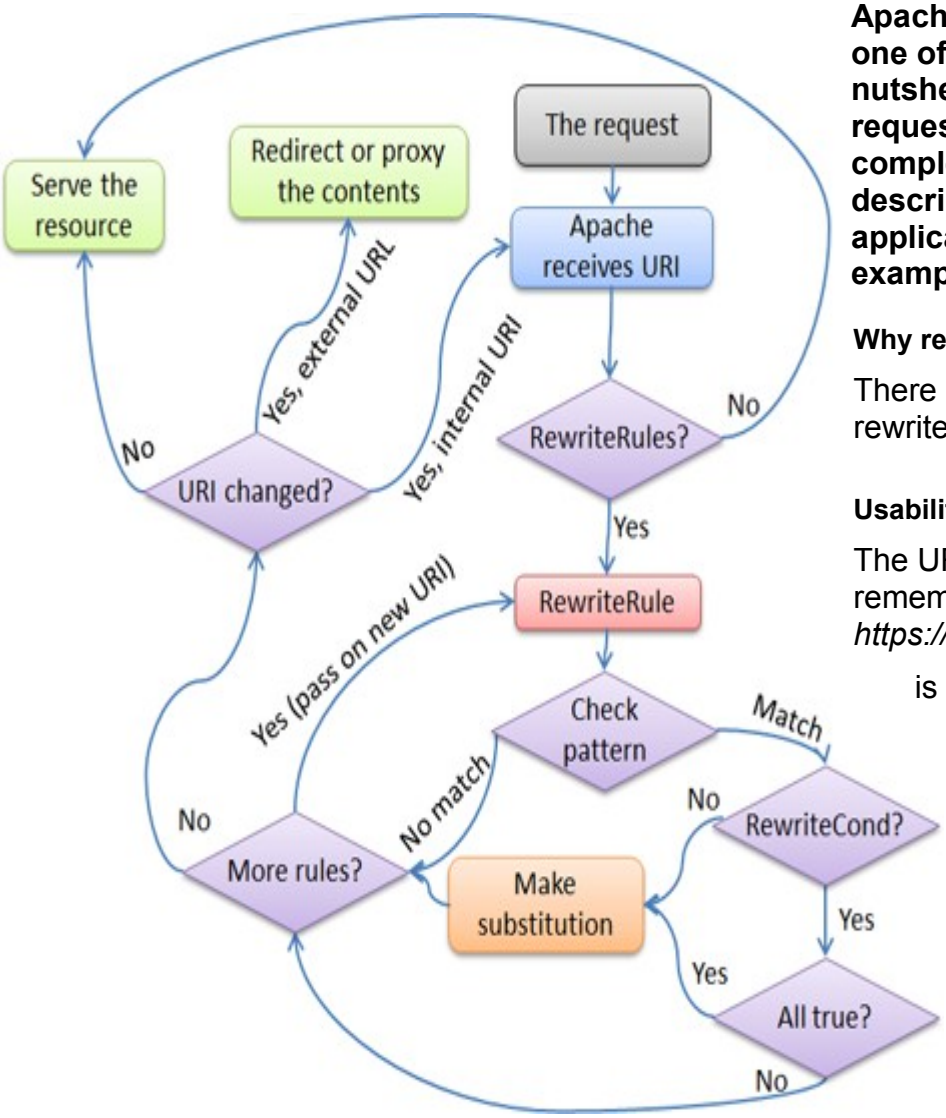
    The SitePoint PHP blog has a new post today about the challenges of password hashing and some of the common risks that can come with it. It's a continuation of a previous article about the actual techniques for hashing in PHP.

15. **Coding the Architecture: Five things every developer should know about software architecture**
    http://www.phpdeveloper.org/news/20867

    While not specific to PHP, this new article on the Coding the Architecture blog gives some good insights on what developers should know about software architecture.

Apache

Rewrite Module



Apache's rewrite module is, arguably, one of apache's most useful features. In a nutshell, it allows you to alter the requested URL to read something completely different. This article will describe why we do this, what applications mod_rewrite has, and examples of getting it to work for you.

Why rewrite?

There are many reasons why people want to rewrite URL's:

Usability

The URL is more readable and people might remember it. A URL like *https://bendechrai.com/contact/*

is easier than

https://bendechrai.com/index.php?page=contact.

Search engine friendliness

Search engines don't like URLs with lots of query-string parameters. The problem with these types of URL's is the search robot might get caught in endless loops of horrible twisty web pages, all cross linking, and all with, essentially, the same page name.

Friendly URLs are search engine friendly, and if they're search engine friendly, you'll get a better search engine position.

Future proof

bendechrai.com used to be purely HTML. Now it's PHP. Next year it might be something else. Why change your URLs every time you change your technology. For years now, https://bendechrai.com/contact/ has pointed to my contact page, and if I change that, then I kill people's bookmarks. The article Cool URIs don't change by Tim Berners-Lee of the W3C might be of further interest.

What to rewrite?

Applications for mod_rewrite are wide and varied, and here is just a sample of uses I've seen that used, or should have used mod_rewrite.

Shopping carts

Too often, I've used a shopping cart and ended up looking at

http://somecart.com/shopping/1009/a-54ff4/?id=41jga

Wow – that tells me a lot. That's the type of URL I see whether I'm looking at a product, my cart, giving my credit card details, or any number of other page types.

Why not have:

http://somecart.com/products/internet/routers/nb1300/

http://somecart.com/yourcart/

http://somecart.com/yourcart/checkout/

etc.

Article systems

As mentioned above, article systems lend themselves well to mod_rewrite. Look at this URL

http://phpmelb.org/content/view/22/48/

Now tell me what that article is about, without clicking the link. Hard, isn't it.

How about:

http://phpmelb.org/articles/apaches-rewrite-module/

A bit easier to understand, I reckon.

Blogs

Okay, maybe I'm flogging a dead horse here, but for those of you who are new to URL rewriting, I'm trying to get as many examples in as possible.

A popular online blog system uses the following URLs:

http://blog/users/username/
http://blog/users/username/year/
http://blog/users/username/year/month/
http://blog/users/username/year/month/day/

In order, these show you the most recent entries; the most recent entries for the year; all entries for the month; all entries for the day. Simple, easy, readable, understandable.

Another blog system has the following URL:

http://site/blog-post-view.php?id=157489

Okay, competition time again: who wrote that blog and when? Yup – no idea either.

How to rewrite?

"Wow – that's cool. But how do I get it to work??"

Woah there, steady. We're getting to that bit. First I'm going to show you how to get it running and an easy way to test it. For the purposes of this article, I'll be assuming you have my set up (Apache 2.x or better on Linux with PHP 4.3.x or better compiled into Apache). Your setup might vary, and while all of this information will probably work, it may not. Head over to our mailing lists or contact us if you have any problems.

Okay, so – installing the module. It's probably already there – all you need to do in your httpd.conf is find the lines that say (they won't be next to each other):

#AddModule mod_rewrite.c

#LoadModule rewrite_module modules/mod_rewrite.so

and remove the # from both, save changes, restart apache and check there are no syntax errors. Fix any errors now to save headaches later.

Okay, so that's all working, head back into your favorite text editor and flick down to the end of the file and put this little snippet in:

```
ServerName test
DocumentRoot /www/test
RewriteEngine On
RewriteRule ^(.*)$ /index.php [L,NS,QSA]
```

What's all that?

Okay, step by step. We've just added a new virtualhost in your apache config file (presuming you're configured for virtual hosts this should all work properly) that handles any requests for the Fully Qualified Domain Name (FQDN) test. We've switched on the RewriteEngine for this virtual host, and created a rule that states that for any request that has zero or more characters of any type between the start and end (in other words, for all requests) rewrite the request to /index.php.

The part in [] at the end means that if successful, this should be the Last rewrite performed, **N**o **S**ubrequests should activate this rule* and that any **Q**uery **S**tring should be **A**ppended.

In the words of apache:

This flag forces the rewriting engine to skip a rewriting rule if the current request is an internal sub-request. For instance, sub-requests occur internally in Apache when mod_include tries to find out information about possible directory default files (index.xxx). On sub-requests it is not always useful and even sometimes causes a failure to if the complete set of rules are applied. Use this flag to exclude some rules.

So in effect, any request will now be redirected to /index.php and any querystring will be maintained. So now all we need to do is make sure that index.php exists and reload apache.

Now before you head over to http://test/ we need to make sure test points to your server! Edit your hosts file* and add the line:

127.0.0.1 test

*Note: The hosts file is usually found at C:\WINDOWS\hosts, C:\WINNT\system32\drivers\etc\hosts or C:\WINDOWS\system32\drivers\etc\hosts on windows systems, and /etc/hosts on most *nix. Some windows setups don't have one, but may have a hosts.sam (sample) file. Just rename this to, or create a new hosts file. Disable the "hide common file extensions" setting in windows explorer options, otherwise you might end up with a hosts.txt file and that won't work!*

Editor Note: On both Windows and linux, you need administrator rights to edit hosts file. On windows, locate 'notepad' in start menu, right click it and select 'Run as Administrator'. Then open hosts file there to edit it. On linux, you can use any editor command with 'sudo'; For example: sudo vi /etc/hosts

Now restart your browser completely – this forces a reload of the hosts file – and head over to http://test/ – it should return a blank page.

Now try http://test/some/non/existant/directory/?id=1 – that should also return a blank page. If it doesn't, you've done something wrong.

But what's the use of a blank page?

So – not very useful. Lets put something in the index.php – here's a cut and paste example:

```
echo 'REQUEST URI = ' . $_SERVER['REQUEST_URI'];
echo '<br>SCRIPT NAME = ' . $_SERVER['SCRIPT_NAME'];
echo '<br>QUERY STRING = ' . $_SERVER['QUERY_STRING'];
```

Output of the script for above URL will be:

```
REQUEST URI =  some/non/existant/directory/?id=1
SCRIPT Name =  some/non/existant/directory
QUERY STRING = id=1
```

Et voila – that's it. Your index.php now knows everything it needs to know to output what you need. Here's just a little code to get you going a bit further:

```php
// REMOVE LEADING AND TRAILING SLASH
$path = trim($_SERVER["SCRIPT_NAME"], '/');

// SPLIT PATH INTO ARRAY
$patharray = explode("/", $path);

// INCLUDE RELEVANT PHP SCRIPT ($patharray[0] is the first directory)
include ($_SERVER[DOCUMENT_ROOT].'/'.$patharray[0]);
```

The included page (some.php in the above URL example) would then have $patharray already and can use that to pull out relevant data from whatever sources you have.

Note: Remember that the URL is now user input – so validation is a must. If someone requests http://test/..%2F..%2F..%2F..%2F..%2F..%2Fetc%2Fpasswd, they might just get a list of all your local user accounts!

This article was written by Ben Dechrai and originally published on the Melbourne PHP Users' Group web site. It is republished here with permission.

Storing Password securely

If you are PHP developer, you must be managing user session on your websites. Managing user session include, managing user password.

For user, password is like a key to unlock features provided by your website and ensure his data will not be accessed by anyone; not even to you as site admin or developer. Imagine a hacker somehow get access to your database, should he have access to password of all users?

Wait a minute, did I heard anyone saying "Why the hacker will hack my site and database? I just run small website without financial or sensitive information."

Well hackers are still interested in you for two reasons:

1. You may be a soft target.
2. Many users have same username/password for multiple website. Once hacker get email/password combination from your site, he may try it on other sites with sensitive information. Yes here user made mistake by having same password for multiple sites but you still can't get away from your responsibility as developer. As developer, you must secure interests of your users, who trust you.

So the password stored in the database must be secured. Fortunately most developers now understand that and encrypt password. If you are not doing that, please slap yourself as hard as you can so that next time you do not repeat the blunder of storing plain password in the database.

Well but encryption is not sufficient. If a password can be decrypted by you, hacker can also decrypt it. Remember if hacker have access to your database, he must be having access to your config and code files and it not difficult to know how to decrypt password. What is the solution; One way encryption?

Still many developers are using md5, sha1 or similar one way encryption function to hash password. One of common example

```
//Saving password
//Get password from user (Through registration form)
$password = getUserPassword();
$hash = md5($password);
//or
$hash = sha1($password);
//Store hash in database.
```

For years, developers used one way encryption. Hackers then start maintaining hash-password mapping database. Take the hash, match that against hash in database and get the password.

To counter that, developers started adding salt to the password. Same password will have different hash and hackers start using 'brute force' attack.

What is brute force attack?
Problem is, both md5 and sha1 are very fast. Hackers can simply generate all possible combination of password, encrypt it and match with hash. Check http://www.braindisorder.org/2008/11/md5-brute-force-with-php/ for brute force example code.

So isn't there any secure way of storing password?
Well there are but only till hackers find loop-holes in best known secure way. Md5 (along with salt) was considered secure for long time but now security loopholes are detected in md5 and brute force is always there. So it is a continuous game. Developers design secure ways and hackers break them. Only way to remain secure is, remain up to date with latest happening in technical world and always use latest security guidelines.

After vulnerabilities were identified in md5 and sha1, community started working on more secure ways. Latest guidelines to store password securely is to use password_hash() and password_verify() functions introduced in PHP 5.5. Here is the example:

```
//Registration - saving password in DB
$password = 'some_password';
$hash = password_hash($password, PASSWORD_DEFAULT);
// We can save this hash in database

// Login - Verifying password
// $password - password supplied by user through login form
// $hash - password stored in database
$verified = password_verify($password, $hash);
// $verified = true or false
```

Second parameter of password_hash allow us to select encryption algorithm. However, as of now, PHP support only one encryption algorithm that is 'bcrypt', which is currently one of the most secure encryption algorithms.

Aah, I don't have PHP 5.5
That's unfortunate. I personally recommend to upgrade to PHP 5.5 but if it is not possible immediately, Anthony Ferrara has back-ported native PHP 5.5 password library to PHP 5.3 and PHP 5.4. Please check it at https://github.com/ircmaxell/password_compat

Kapil Sharma is Tech Lead at Ansh Systems. He is professionally involved in web development since 2004 and have major interest in Unit Testing, web security, refactoring, scaling web applications etc.

Kapil is one of the founding members of PHP Reboot and regular speaker in PHP Meetups.

Twitter: kapilsharmainfo

website: www.kapilsharma.info

Blog: blog.kapilsharma.info

PHP Myth

PHP is an old language. Its development started in 1994. At that time, it was just few CGI (Common Gateway Interface) binaries written in 'C' language to make dynamic pages for its author Rasmus Lerdorf.

Again, internet now is completely different then what it was in 1994. Although PHP matured with time, provide all the tools needed by developers, it have a long history. During all these years, PHP become on one of the most widely used languages to design web applications. Millions of developers used PHP; some like it and some hate it.

Those hate it, gave their reasons why they hate it. No thing is perfect in the world and yes, PHP too have some weak points. PHP learned from its mistake and fixed most of the weak points but being old language, those points are still criticizing PHP. So now, they can be considered as PHP Myth.

In this article, we will go through some of those common PHP myths and find if they are still valid; **Fact or myth**. Again, if it is a fact, we will try to figure out why latest versions of PHP had not fixed them.

So let's us start by listing some of the common myth around PHP.

- **PHP is crappy and there are hell lot of crappy PHP programs.**
- **PHP is good only for web development.**
- **PHP is interpreted language so it is slow.**
- **PHP mix business logic and presentation logic.**
- **PHP is insecure.**
- **Development in PHP is slow.**
- **You cant scale PHP application well.**
- **You need to buy from Zend to make PHP work best.**
- **It is not good at OOPs.**
- **PHP in not good/sufficient for serious programmer.**

PHP is Crappy and there are hell lot of crappy PHP programs.

Let me break this point into two. First **PHP is crappy**; well it depends on what is your definition of crappy. Most of the points we are discussing in this article falls under crappy definition. So to answer if PHP is crappy or not, lets go through remaining PHP Myths.

Second, **there are lot of crappy PHP programs,** is actually fact. Reason is, PHP is supposed to be very simple, and easy to learn. Now if it is easy, there are lot of amateur programmers (or PHP users). Most of these amateur PHP users just go to some online PHP tutorial (which could be very old) and try to make some dynamic feature of their website. Can we expect them to write high quality, professional code? Obviously not and they add some of crappy PHP code.

Again there are few professional PHP developers, with limited experience and little learning curve. They either become freelancer or gets job in lot of small software companies, which work on simple dynamic websites. Can we expect high quality code form them? No.

Not there are mid level programmers. They have ability to write high quality code but they prefer to do things which **seems easy**. They also add crappy PHP code. One of major goal of PHP Reboot is reboot these programmers and encourage them to write better code.

I also met many PHP programmers who use PHP to write modular, reusable code with proper separation of logic and backed by unit test cases with 100% code coverage of business logic. Yes they reach at level where we cant call them 'PHP programmer' but 'Programmer'.

So even though second point is fact, it is fact with any available programming language. Crappy code depends on programmer, not programming language. PHP is well capable of produce code which can not be bettered in any other programming language.

PHP is good only for web development.

That a fact but when did PHP claimed otherwise. PHP, right from beginning was designed to make web applications. If you want to write desktop application or native mobile application, PHP is not designed for that.

Is it a negative point? Not for me. PHP does what it is supposed to do and do it very well.

Take an example of wordpress, written in PHP. I don't want to claim any numbers but it is far ahead of any other open source (or proprietary) tool that power websites. I personally don't like wordpress as I feel it could be written in better way but does it matter? It simply do what it is supposed to do and do it very well. This is the key to success.

PHP is designed to design web applications and it provide all the tools you need; regardless you make single page website or enterprise application.

PHP is interpreted language so it is slow.

Fact. Yes PHP is interpreted language and interpreted languages are slower then compiled languages.

PHP is sloe; myth.

PHP was designed to be fast. Even though it is interpreted language, it is fast. Here too, programmer need to use right tool with performance in mind while coding. On server, there are lot of accelerators available for PHP. An accelerator cache opcode of PHP script and next time same page is requested, it behaves much like compiled language.

Again with latest PHP 5.5, OPCache PHP extension become part of PHP core and OPCache is one of the bast opcode caching solutions available.

PHP mix business logic and presentation logic.

Myth.

It is possible to mix business logic and presentation logic in PHP but it is also possible to keep them separate.

Again tell me a single programming language where we can't mix them. I started my career as Java developer and know we can mix business logic and presentation logic in JSP or servlet. During my career I also worked with Python and perl for some time and it is possible in these language as well. NodeJS is recently getting popular and we can mix them in NodeJS too.

So finally it depends on developer. If you want to make your code crappy, you can do it in any language. At the same time, you can write very good code in PHP or any other language.

PHP is insecure.

This is one of the biggest myth about PHP. Let me take that subject a bit in details.

To start off, no thing is 100% secure over internet. Also, PHP provide all the tools to write secure web applications, as secure as possible in any other language. In the coming issues of PHP Reboot, we will go through web application security best practices in details.

However, yes PHP did some mistakes in the past, which are not valid now a days. One of the biggest mistake was 'register globals'. Again, it was not a security vulnerability but yes, a security risk. Lets understand it in bit details:

'Register globals' were intended to make programmers life easy. It take all the parameters passed in a request and set them as global variables, that programmer might use. So what was the issue? There was no issue unless programmer make a mistake. Consider following program, some of you might be familiar with such program.

```
If ( isAdminUser() ) {
    $admin = true;
}
if ( $admin ) {
    //load admin panel
}
```

Is there anything wrong with this program? There is a basic coding best practice, 'NEVER TAKE IMPORTANT DECISIONS BASED ON A VARIABLE THAT MIGHT NOT BE INITIALIZED.' Here if isAdminUser() returns false, $admin will not be initialized. Did you get the problem now?

If register globals 'on', a hacker can use URL 'http://domain.com/page.php?admin=1' and register global will set $admin = 1. Did you see the problem now? Yes with above code and URL, a hacker could bypass admin authentication and can get access to admin panel.

Was it problem of register globals or programmer? Obviously programmers problem but register global contributed to that. PHP realized that many years ago and register globals was turned off by default in PHP 4.2.0 and any version that came after that. Also it will be completely removed from PHP version 5.6.

But that single instance hurt PHP reputation badly. PHP 4.2, which fixed that problem was released in 2002, approximately 12 years ago. So now, we can easily say PHP is secure and this point is completely myth.

Development in PHP is slow.

One of the common PHP myth is that it take considerable time to develop web applications in PHP. There are programming languages which take more development time then PHP like java. I don't know source of this myth but my guess, it came form Ruby on Rails and/or Django (Web framework of python)

But this is not a fair comparison. Let me ask you a question.

If I have 10 pen and I give you 3 pencils, what is remaining with me?

Ruby is a programming language but rails is a framework. We can't compare programming language with framework. If you want to compare with Rails and Django, compare them with many of available PHP frameworks; Laravel, Symfony, Yii, Cake PHP etc. Yes many of them are inspired by rails but they are available in PHP world.

So this is a pure myth.

You can't scale PHP application well.

I want to take with some statistics. Facebook get nearly same number of hits as Google. It serves several thousand user request at any given instance of time. Facebook now moved to Hack on HHVM but earlier it was on PHP. It is the biggest proof that there is no scaling issue with PHP.

Yes scaling ay web application need special skill set but as far as possibility is concerned, there is no scaling issue with PHP. MYTH.

You need to buy from Zend to make PHP work best.

Zend is the company behind PHP and has lots of their stakes in PHP. Still Zend is not dominant or only solution provider of any thing related to PHP. If you don't like Zend products, there are other options available.

So this too is a myth.

It is not good at OOPs.

Few years ago, it was a face but not any more. PHP took serious steps towards object oriented programming with PHP 5.0m which was released in 2004. Now PHP support nearly all OOPs principles. So this too is a myth.

PHP is not good/sufficient for serious programmers.

PHP contributed maximum numbers of web based open source projects. Some of them are very dominating in their respective field like Wordpress, Joomla, MediaWiki (Wikipedia), Megento, SugarCRM etc. Were the programmers wrote those tool not the serious programmers? In my office, I work with many serious PHP Programmers and communicate with many over internet.

I'll not call it only a myth but a rubbish statement.

So is everything right with PHP?

Although there are lot of PHP myths, there are many valid concerns as well. However lets keep that discussion open for some other day.

For now, I just want to add, even with all the myths and critics, PHP is one of the best programming languages for writing a web applications.

Kapil Sharma is Tech Lead at Ansh Systems. He is professionally involved in web development since 2004 and have major interest in Unit Testing, web security, refactoring, scaling web applications etc.

Kapil is one of the founding members of PHP Reboot and regular speaker in PHP Meetups.

Twitter: kapilsharmainfo

website: www.kapilsharma.info

Blog: blog.kapilsharma.info

So you want to write test

I often get asked for some advice on how to get started with writing tests for your PHP code. It's a fair question, since I am presenting myself as an expert-ninja-rockstar-sensei-opinionated-egomaniac on the topic. I often struggle with coming up with an answer that can fit into the 140 characters available via Twitter, but clearly this is not a good strategy.

To all my loyal followers, who put up with my never-ending stream of nonsense and provide a slowly-increasing portion of my income, here are my thoughts on how to get started with testing your PHP code.

Learn how to recognize untestable code

The worst feeling in a programmer's world has to be sitting in front of a piece of code that you didn't write yourself but is critical to the success of your business. You will be told that it's of the utmost importance that you GOD SAKES DON'T BREAK ANYTHING OR ELSE WE ARE DOOMED.

While I don't get the same kinds of warnings from my employers, I am familiar with that feeling of trying to figure out not only what the code in front of you is supposed to do, but what the developer who came along before you was even THINKING ABOUT in that code.

When it comes to testing code, there are two principles that you have to strive to aim for:

- **find a way to pass dependencies into your code**
- **find a way to break down your code into the smallest modules possible**

By following this two simple-but-extremely-difficult-to-do ideas, you will end up with code where writing tests becomes so easy you will be doing it without a second thought.

No more static method calls. Get rid of those private and protected methods. Stop creating new objects inside a method. Think about passing data around instead of objects. Work hard to make sure your methods have little-to-no side effects). Decouple your data source from the code that manipulates results from it. More importantly, understand why all these things are barriers to having an easily-testable code base.

In short: do all the kind of hard work and drudgery that many managers tell you is a waste of time and will not return any value to your employer. I say screw those guys and instead feel awesome about working on building light, nimble systems full of tests that let you go home from work on time.

Keep learning the language

Tests are just code. Keep learning how to write better PHP code. Understand why so much of what is on PHP Sadness is actually trying to help us get better. Understand why the array in PHP is awesome. Learn the iterator drinking game.

In short, the more you know about the language the more likely you are to learn how to solve problems using core functionality instead of constantly implementing solutions in code you wrote yourself.

Chain units of code together for greatness

I've talked about in other places about my thoughts on the UNIX philosophy. For the impatient, it is about creating small, single-purpose units of code and then chaining them together to accomplish awesome things.

If you're ever searched online for ways to do things from the command line in various flavours of Linux distributions, you will understand what I mean. "Take sed and cat and awk and grep and pipe it through cut and then to sort and you will get exactly what you want."

That sort of stuff is awesome, and fits in rather nicely with the concept of writing small modules of easily-testable code.

Start asking "how am I going to test this?"

This is the first question I ask when I am presented with a problem that I need to solve using code. From that one extremely important question all the best practices that are associated with building highly-testable code emerge from.

If it is not obvious how you are going to be able to create automated tests for a particular set of functionality, it probably means you haven't thought hard enough about the problem you are trying to solve.

I'm not saying every problem has a testable solution. But any problem where you have expected results based on known input is a scenario that you can test for. Database queries. 3rd party API's. Request routing. Fizz Buzz. These are all things that we can test, by making sure to follow some of those key ideas I advocated above.

Test-centric development is about using tools as a way to create API's and interfaces that are simple, easily testable, and easy to modify going forward.

Adding new features becomes a potentially-simple process consisting of:

- **design new feature**
- **write tests for these new features**
- **write code until the tests pass**
- **go home on time**

Stop people from pushing code without proof it's fixed

Sure, you don't have any tests RIGHT NOW. But I'm sure you've got bugs to fix in your code. Write tests to verify that the bug exists. I'm sure you'll find your code isn't as modular or testable as you thought it was.

Once you've reproduced the bug, then write some code until the bug GOES AWAY. Then push your updated code into production. Rinse and repeat until you have a very highly-targeted test suite and a bunch of highly-targeted fixes for known mistakes in the application. That is what I call real-world testing.

So make sure that nothing that fixes something goes up into production, no matter how desperate the circumstances, without proof that this fix actually is a fix instead of a wild guess.

Stop people from doing things manually

It's the 3rd year of the 2nd decade of the 21st century. The internet is full of instructions on how to automate just about anything. Spend an afternoon studying that instead of dreaming about an advertising-driven disruptive startup.

Chef. Puppet. Shell scripts. Jenkins. Travis. Virtualized servers. Learn these things. Use them. You will thank me. Maybe enough to send me money via PayPal. Automate everything that you keep finding yourself doing manually, and then figure out if there are ways to not even do those things. You might be surprised.

Stop using tools without test suites

I treat any PHP tool that does not have its own test suite as a black box that cannot be trusted. Tests show that the developer cares about the quality of the work they do, and also provides real examples on how to do things.

Black boxes are not things you should be relying on when the going gets tough. Treat any code that you cannot easily test as being suspicious and likely broken. I feel the same way about code *I* have written, so imagine what I must think of other people's stuff.

Always wonder if you're doing it right

Integration tests are a scam. Unit tests don't fix bugs. Simple enough code doesn't require tests. Pair programming is a cargo-cult practice. PHP sucks. Mobile applications are the future. Documentation comes after the work is done. Nobody likes a developer who asks too many questions. You work in the real world where results matter. Exit statuses are better than throwing exceptions. Asynchronous calls always result in callback hell.

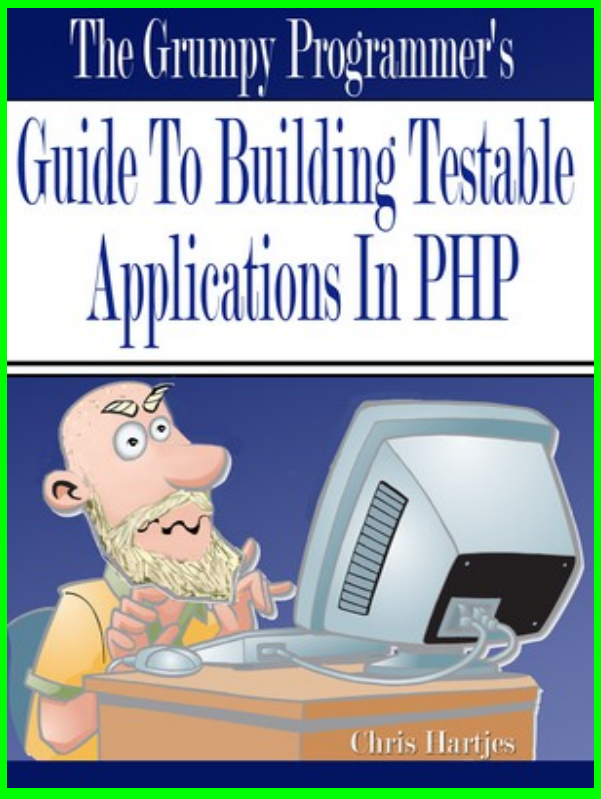
Not everything in the above paragraph is true. The only way to find out is to get out there and figure out if it is true or not for yourself.

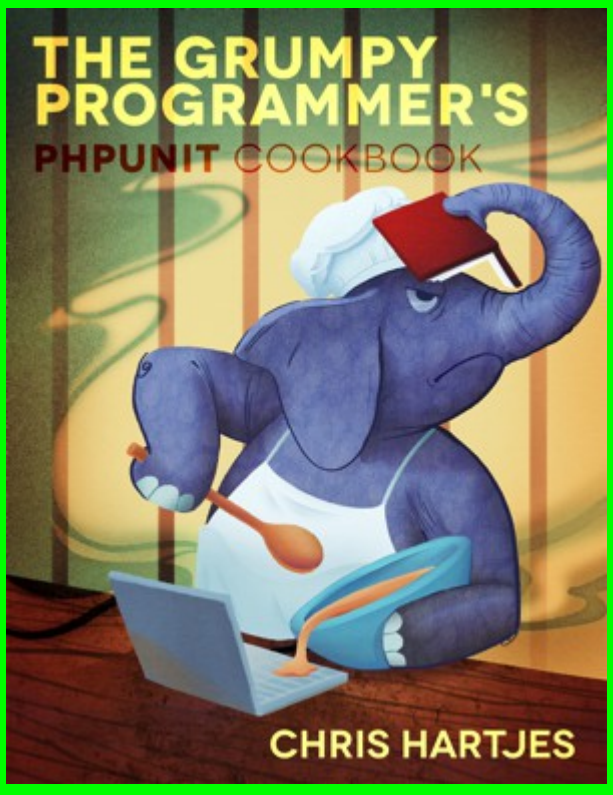
Keep on testing. I'm here to help.

This article was originally posted by Chris Hartjes on his blog www.littlehart.net. It is republished here with permission.

Chris Hartjes is one of most popular names in international PHP community. He is one of the biggest advocate of writing unit tests for your application. His blog is a good source of many high quality articles about unit testing. If you are not writing unit tests for your application, we recommend to read his blog posts once.

Chris also wrote two books on testing:





Twitter: @grmpyprogrammer

Some of the topics we shortlisted for next month are:

Introduction to Hack and HHVM

HHVM is the virtual machine to execute PHP and Hack programs. Hack is a programming language, similar to PHP with few major changes like static typing.

In this article, we will setup HHVM on development machine and make a simple application.

Introduction to Laravel

Laravel is one of the most popular and best designed PHP frameworks available. In this article, we will set up laravel locally and make a small web application using Laravel.

PHP Reboot: PHP 5.3

This is the first part of series of articles where we will go through recent changes in PHP. In first part, we are going to concentrate on changes in PHP version 5.3

Session management in PHP

Nearly all PHP application need to provide user login and once user login, they need to manage user session. This article is all about how to manage PHP sessions securely.

Web security

PHP is used to make web applications and on internet, one of the most important concern is security. In this article, we will figure out some of the most common security threat and how to secure our applications against those threats.

There are few more articles to be decided for next issue of PHP Reboot.

Based on your experience, if you have any article to be shared with PHP community, please write us as phpreboot@gmail.com. We will convert your experience in an article and share with the community.

Pune meetups in April 2014

There are few good PHP and related technologies meetup planned in April. Some of them, we figured out at the time of writing are:

Drupal Camp

Sunday, April 5th, 2014

Symbiosis Institute of Management studies, Range Hills Road, Khadki, Pune.

Two days packed with great interactive sessions about #Drupal in four tracks: DevOps, Front-end and Mobile / Responsive Design, Business & Marketing Cases. Drupal in a Day training. Workshop for companies and entrepreneurs. CXO roundtable. Showcase Openspace and Birds of Feather. QA with rock star devs to help with your Drupal projects. And a great After Party to wind down and make great new friends!

Register at http://camp2014.punedrupalgroup.com/user/register

Lets talk wordpress

Saturday, April 5th, 2014

Cafe Coffee Day, Kalyani Nagar, Pune.

Let's discuss our love for WordPress and how we can contribute back to community

RSVP at http://www.meetup.com/Pune-WordPress-User-Group

PHP Reboot: PHP Myths and best practices.

Saturday, April 12th, 2014

Equal Experts India Pvt. Ltd. 4-C, Cerebrum IT Part, Kalyani Nagar, Pune.

Whole purpose of PHP Reboot is to learn/share new and advanced features of PHP and supporting technologies.

In this first session, we are trying to decide future path. So we will have following agenda for our first meetup:

1. PHP Myths (Discuss about issues with PHP; Real and advertised)
2. How latest changes in PHP overcome them?
3. What are best practices and why it is important understand and follow them.
4. A little introduction to the topics we will discuss in future.
5. What exactly local developers expect from local PHP community. This will be open session to decide the future of PHP Reboot community.

RSVP at http://www.meetup.com/PHPReboot

Available to download at

PHPReboot.com

For latest news
follow us on twitter
@phpreboot

or Facebook
www.facebook.com/phreboot