Jumping into PHP 7

// Dallas PHP - March 2016

500

Internal Server Error

EXPLOSION

What you might be doing wrong right now.

Flat Out Gone

ereg	preg_match
call_user_method[_array]	call_user_func[_array]
set_socket_blocking	stream_set_blocking
all postscript 1 functions (imageeps*)	migrate to TTF for rendering fonts
$HTTP_RAW_POST_DATA	file_get_contents("php://input")
<%, <script language="php">	<?php // :p

Extension: MySQL - Jump ship and migrate to PDO... or be lazy and migrate to MySQL Improved by adding "i" to all your functional calls... mysql_connect => mysqli_connect. Strongly consider restructuring for OOP.

Extension: MSSQL - Migrate to PDO or SQLSRV.

Error and Exception Handling

A lot of things that used to throw fatal errors now throw catchable Errors. You are now able to catch a lot more things than you were able to previous.

May seeing some uncaught errors flying around afterwards.

Fatal Error: Cannot declare class Error.

Error and Exception Handling (cont.)

interface Throwable;

- Exception implements Throwable;
  - Exceptions as usual. You should not notice [m]any differences with your existing error handlers for past exceptions.

- Error implements Throwable;
  - More kinda serious exceptions thrown by the engine instead of the old errors that you could do nothing about but laugh at.

```php
try {
    require('include-with-syntax-error.php');
}

catch(Exception $Error) {
    echo "Something Failed:", PHP_EOL;
    echo $Error->GetMessage();
}

catch(Error $Error) {
    echo "Oh Freaking Snap:", PHP_EOL;
    echo $Error->GetMessage();
}
```

```php
try {
    require('include-with-syntax-error.php');
}

catch(Throwable $Error) {
    echo "Whatevs Attempted Ragequit:", PHP_EOL;
    echo $Error->GetMessage();
}
```

Bane of E_STRICT

E_STRICT still exists, but it is currently empty. All things that used to spam E_STRICT things have been "downgraded".

- E_DEPRECATED for old things.
- E_NOTICE for dumb things you did.
- E_WARNING for dumber things you did.

Changes to Dynamic Variable Resolution

Variable resolution is now left-to-right precedence. This means you WILL experience problems if you were not very explicit with your magical magic. This also means they are going to read really weird.

Solution: Always Be Explicit... and TBH... just avoid this feature unless it is super clever.

$$Val['Key']

${$Val['Key']}	($$Val)['Key'];

$Object->$Property['Key']

$Object->{$Property['Key']}	($Object->$Property)['Key'];

Changes to Yielding (Generators)

Yield is now a right-to-left precedence keyword. Some things that read really odd before should read a bit nicer.

Yield can now also yield another generator now to extend its iteration train.

Yield can now also return a final value after-the-fact, retrievable by the GetResult method.

echo yield -1;

echo (yield) - 1; // wat

echo yield (-1); // thur we go

```php
function TrainEngine() {
    yield "\u{1F682}";
};

function TrainPayload() {
    for($Iter = 0; $Iter < 2; $Iter++)
    yield "\u{1F683}";
};

$Train = (function() {
    yield from TrainEngine();
    yield from TrainPayload();
    return 'Choo Choo I ams an Trains';
})();

foreach($Train as $Val)
echo $Val;

echo PHP_EOL, $Train->GetReturn(), PHP_EOL;
```

```
> php yield\train.php

🚂🚃🚃

Choo Choo I ams an Trains
```

Changes to FOREACH()

No longer moves the internal array pointer.

Now operates on a "copy" of the array while doing *by-value* looping (while still seeming to respect copy-on-write behaviour). This means appending during foreach() will not result in an extended loop.

Appending during foreach() while doing a *by-reference* loop will now extend the loop consistently/properly.

https://gist.github.com/bobmajdakjr/7553df62287cd534f31a

```php
$Data = range(1,10);

foreach($Data as $Key => $Value)
$Data[] = $Value;

printf(
   'Key: %d, Value: %d%s',
   key($Data),
   current($Data),
   PHP_EOL
);

print_r($Data);
```

```
> php foreach\lol-by-value.php

Key: 0, Value: 1
Array
(
    [0] => 1
    [1] => 2
    [...]
    [18] => 9
    [19] => 10
)
```

```php
$Data = range(1,10);

foreach($Data as &$Value)
$Data[] = $Value;

printf(
   'Key: %d, Value: %d%s',
   key($Data),
   current($Data),
   PHP_EOL
);

print_r($Data);
```

```
> php foreach\lol-by-ref.php

Fatal error: Allowed memory size of 134217728 bytes
exhausted (tried to allocate 134217736 bytes) in lol-
by-ref.php
```

Changes to LIST()

The list() construct is no longer able to split strings for you. Use str_split.

The order of items has been normalised when appending onto an existing array with the []'s in the list.

- list($A[], $A[], $A[]) = [1, 2, 3];
- PHP 5: $A = [3, 2, 1];
- PHP 7: $A = [1, 2, 3];

Changes to JSON handling

The JSON Parser was changed under the hood which results in a few changes in how it handles floats. Less leeway on how bad you can malform them.

The decimal cannot be the last thing anymore, in normal notation and science notation:

- json_decode('42.'); // NULL
- json_decode('42.e42'); // NULL

Cut the decimal point out or append a zero:

- '42', '42.0'
- '42e42', '42.0e42'

General Changes to Function Definitions

Multiple arguments with the same name are no longer valid and is fatal.

- function Whatevs($Thing, $Trash, $Trash, $Trash) {

func_get_args now tracks the reference of the arguments rather than the value, but only if they were defined by the function signature...

```php
function WhoAreYou($FirstName, $Surname) {
   $FirstName = 'Bob';
   $Surname = 'Magic';

   print_r(func_get_args());

   return "{$FirstName} {$Surname}";
}

echo WhoAreYou('Jean Luc','Picard'), PHP_EOL;
```

```
> php func\func1.php

Array
(
    [0] => Bob
    [1] => Magic
)

Bob Magic
```

```php
function WhoAreYou() {
    list($FirstName, $Surname) = func_get_args();

    $FirstName = 'Bob';
    $Surname = 'Magic';

    print_r(func_get_args());

    return "{$FirstName} {$Surname}";
}

echo WhoAreYou('Jean Luc','Picard'), PHP_EOL;
```

```
> php func\func2.php

Array
(
    [0] => Jean Luc
    [1] => Picard
)

Bob Magic
```

General Data Type Changes (Integers)

Negative bit shifts are no longer valid and will throw an ArithmeticError.

- $Result = 4 << -2;
- $Result = 4 << $CalculatedShift; // :(

Shifting too wide will always result in 0.

- $Result = 1 << 9001;
- int(0)

Division by 0... Prepare your WTFing...

- var_dump(3/0); // float(INF) (and a E_WARNING)
- var_dump(-3/0); // float(-INF) (and a E_WARNING)
- var_dump(0/0); // float(NAN) (and a E_WARNING)
- var_dump(0%0); // CATCHABLE FATAL ERRRRRRRERRRR
- used to just be warnings and bool(false) for all the things...

General Data Type Changes (Strings)

is_numeric( ) no longer claims hex values are numeric.

- "is it generic human understandable"
  - "15" yes.
  - "0xf" no.

- $Int = filter_var('0xf', FILTER_VALIDATE_INT, FILTER_FLAG_ALLOW_HEX);

- $Int = hexdec('0xf');

General Data Type Changes (Objects)

Using the Store-By-Reference operator is no longer valid when instantiating new objects.

- $Object =& new Whatever;

- Objects are already PBR... you do not want to pass an object by reference reference. If you think you do, you don't.

NEW THINGS

Start using these things.

Scalar Type Hints

Classes, Interfaces, and Arrays...

- Integers (int)
- Floats (float)
- Booleans (bool)
- Strings (string)
- The Object Doing It (self)

Unfortunately not properly nullable yet... but...

function Whatever(Type $Arg): ReturnType { }

Scalar Type Hints (cont.)

By default type hints are "coerced" or weak typed.

Files can declare they want super strict typing by adding a declaration at the top of the file.

This (strict types) only affects the file in question and not calls to functions within the file from outside the file.

The only way to allow "something or null" is with =NULL on the arg. There is no valid syntax like "?int" yet to mark it nullable. This also has the side effect of making it optional... which may not actually be desirable...

```php
function
WhoAreYou(String $FirstName, String $Surname):
String {
    var_dump($FirstName,$Surname);
    return trim("{$FirstName} {$Surname}");
}

echo WhoAreYou('Bob','Magic'), PHP_EOL;
echo WhoAreYou(42,42), PHP_EOL;
```

```
> php types\type-loose.php

string(3) "Bob"
string(5) "Magic"
Bob Magic

string(2) "42"
string(2) "42"
42 42
```

```php
declare(strict_types=1);

function
WhoAreYou(String $FirstName, String $Surname):
String {
   var_dump($FirstName,$Surname);
   return trim("{$FirstName} {$Surname}");
}

echo WhoAreYou('Bob','Magic'), PHP_EOL;
echo WhoAreYou(42,42), PHP_EOL;
```

```
> php types\type-strict.php

string(3) "Bob"
string(5) "Magic"
Bob Magic

Fatal error: Uncaught TypeError: Argument 1 passed to
WhoAreYou() must be of the type string, integer given,
called in type-strict.php
```

```
function
WhoAreYou(String $FirstName, String $Surname=NULL):
String {
    var_dump($FirstName,$Surname);
    return trim("{$FirstName} {$Surname}");
}

echo WhoAreYou('Bob'), PHP_EOL;
```

```
> php types\type-nullable.php

string(3) "Bob"
NULL

Bob
```

```php
function
FilterPageNumber(Int $Page):
Int {
    if($Page < 1) $Page = 1;

    return $Page;
}

var_dump(FilterPageNumber(42));
var_dump(FilterPageNumber('42'));
var_dump(FilterPageNumber('adfaf'));

// var_dump((int)'adfaf');
// int(0)... so surely... in non-strict mode...
```

```
> php types\type-loose-suck.php

int(42)
int(42)

Fatal error: Uncaught TypeError: Argument 1 passed to
FilterPageNumber() must be of the type integer, string
given, called in type-loose-suck.php


> f*** me.

Command not found.
```

NULL Coalescing Operator (??)

Is it defined and not NULL? Then have it else have this other thing.

Chainable, reading from left to right.

Similar to the Ternary operator.

It is safe to use on array keys which may not exist, similar to isset().

```php
function GetCurrentPageOld() {
   return (isset($_GET['Page']))?
      ((Int)$_GET['Page']):
      (1);
}

function GetCurrentPageNew(): Int {
   return (Int)$_GET['Page'] ?? 1;
}

$_GET['Page'] = 42;
var_dump(GetCurrentPageNew());

unset($_GET['Page']);
var_dump(GetCurrentPageNew());
```

```
> php other\nullcoal.php

int(42)
int(1)
```

Spaceship Operator (<=>)

Less Than, Equal To, Greater Than all in one.

Returns -1, 0, or 1, depending on the result of the expression. "Troolean"

Allows you to quickly write simple sorts.

```php
$People = [
    new Person('William','Riker'),
    new Person('Jean-Luc','Picard'),
    new Person('Bob','Magic'),
    new Person('Geordi','La Forge'),
    new Person('Thomas','Riker'),
    new Person('Worf','Son of Mogh')
];

usort($People,function($A,$B){
    return
    ($A->GetSurname() <=> $B->GetSurname())?:
    ($A->GetFirstName() <=> $B->GetFirstName());
});

foreach($People as $Person)
echo $Person->GetRosterName(), PHP_EOL;
```

```
> php other\spaceship.php

La Forge, Geordi
Magic, Bob
Picard, Jean-Luc
Riker, Thomas
Riker, William
Son of Mogh, Worf
```

Constant Arrays

Like normal constants. Only... arrays.

- define('SITELIST', ['one','two','three']);
- echo SITELIST[0];

Numerical or Associative, whatevs.

Anonymous Objects

Just like anonymous functions, you can now defined anonymous classes to create anonymous objects.

- $Object = new class { public function Wut() { return 'Wut'; } };
- echo $Object->Wut(), PHP_EOL;

Please um... use... sparingly...

Unicode Codepoint Escape Sequence

You can now generate UTF characters within strings with the \u{} escape sequence. It takes the codepage value in hex.

```php
function TrainEngine() {
    yield "\u{1F682}";
};

function TrainPayload() {
    for($Iter = 0; $Iter < 2; $Iter++)
    yield "\u{1F683}";
};

$Train = (function() {
    yield from TrainEngine();
    yield from TrainPayload();
    return 'Choo Choo I ams an Trains';
})();

foreach($Train as $Val)
echo $Val;

echo PHP_EOL, $Train->GetReturn(), PHP_EOL;
```

Expectations Via Assertions

So assert() was already a thing in PHP, but it has been upgraded.

Allows you to include debugging assertions within the code, with the ability to toggle them off so they have zero performance impact on production.

Allows you to throw exceptions for things you may need to debug, but may not actually be fatal to the process.

This is a multistep process.

Expectations Via Assertions (cont.)

Expectations can be configured to either be Fatal, Warnings, or Silent.

This is done via two different INI directives.

- assert.exception = 0 or 1 (default 0)
  - 0 = Run the assertion but instead of throwing the exception, spew the exception as a Warning
  - 1 = Run the assertion and exception it hard if it fails.
  - Can be set via ini_set().
- zend.assertions = -1, 0, or 1 (default 1)
  - 1 = Compile and Execute the assertion. (dev mode)
  - 0 = Compile but skip the assertion.
  - -1 = Skip it completely for Zero Impact. (prod mode)
  - Cannot be set via ini_set().

```php
$Result = FALSE;

assert(
    ($Result instanceof FilesystemIterator),
    (new Exception('We did not get our iterator.'))
);

echo "Whatevs", PHP_EOL;
```

```
> php expect\assert1.php

Warning: assert(): Exception: We did not get our
iterator.

Whatevs
```

```
> php -d assert.exception=1 expect\assert1.php

Fatal error: Uncaught Exception: We did not get our
iterator. in assert1.php

Stack trace:
#0 {main}
  thrown in assert1.php
```

```
> php -d zend.assertions=-1 expect\assert1.php

Whatevs
```

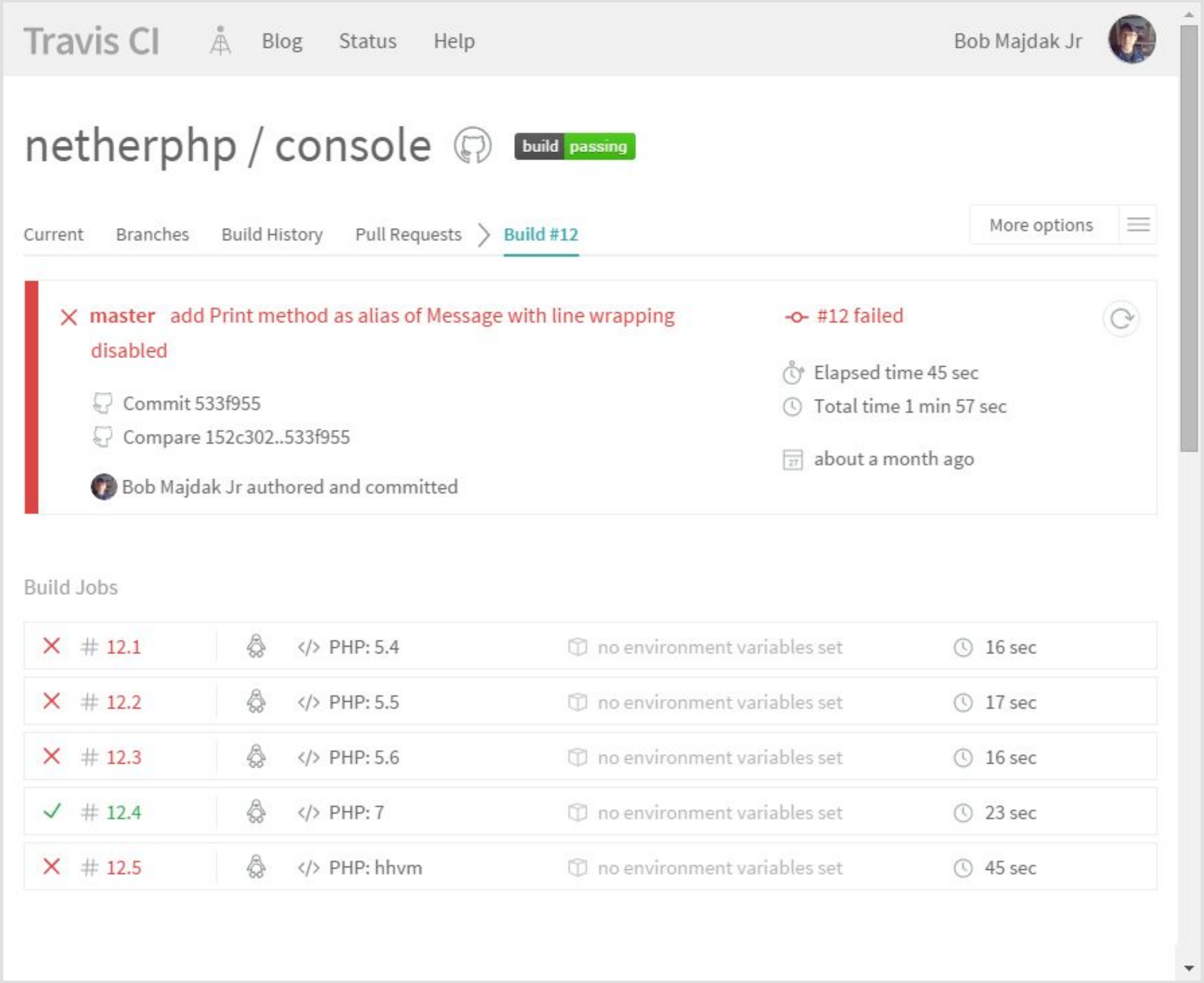
Closure Call Binding

Anonymous functions / closures have had an upgrade to make it easier to rebind the value of $this within the function via the Call method.

```php
$Func = function() {
   $this->Guffaw();
   return;
};

$Func->Call(new class{
   public function
   Guffaw() {
      echo 'roflmao', PHP_EOL;
      return;
   }
});
```

Looser Restricted Word Restrictions



netherphp / console build passing

✗ master add Print method as alias of Message with line wrapping disabled

Commit 533f955

Compare 152c302..533f955

Bob Majdak Jr authored and committed

#12 failed

Elapsed time 45 sec

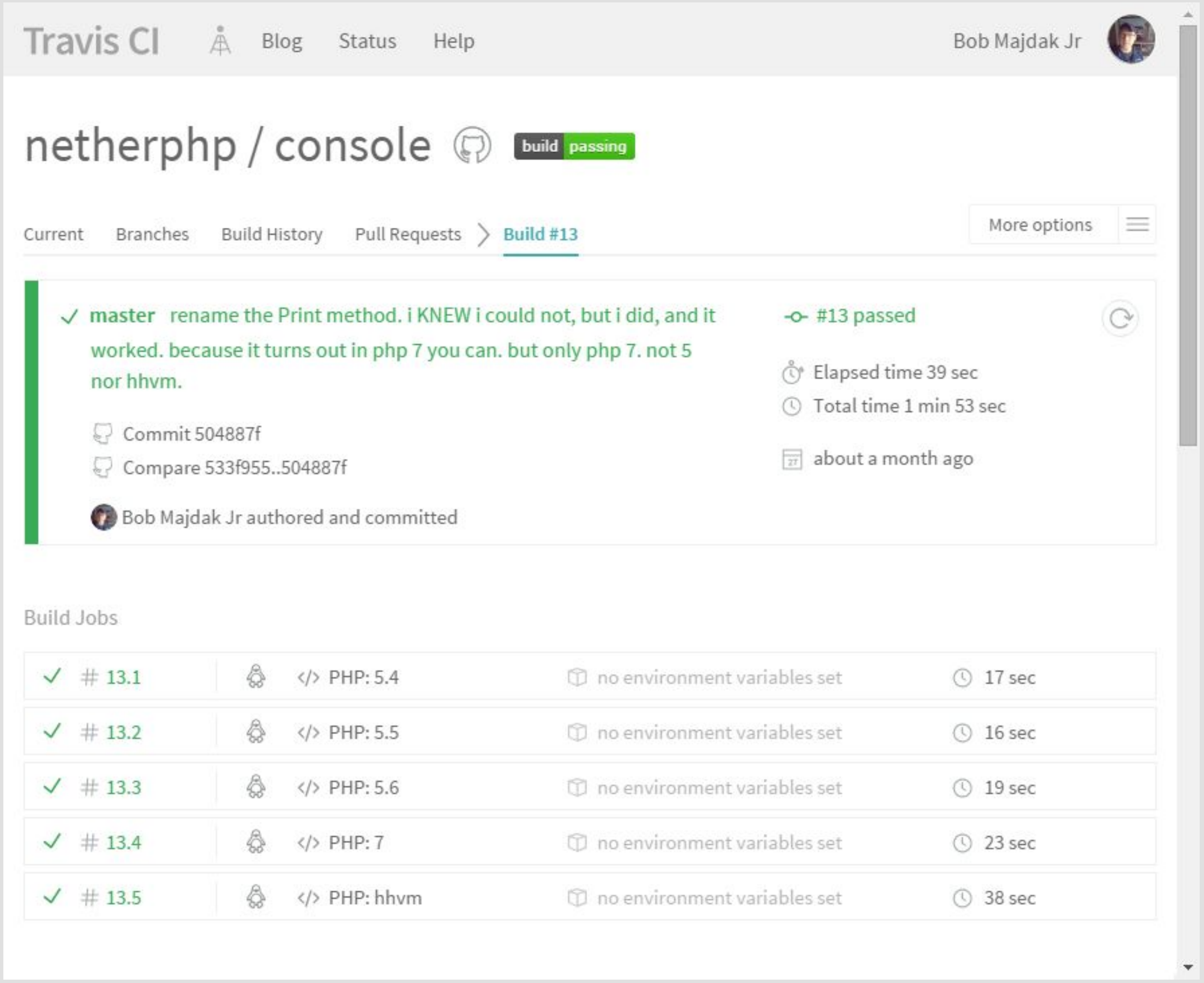
Total time 1 min 57 sec

about a month ago

Build Jobs

✗	# 12.1	PHP: 5.4	no environment variables set	16 sec
✗	# 12.2	PHP: 5.5	no environment variables set	17 sec
✗	# 12.3	PHP: 5.6	no environment variables set	16 sec
✓	# 12.4	PHP: 7	no environment variables set	23 sec
✗	# 12.5	PHP: hhvm	no environment variables set	45 sec

Looser Restricted Word Restrictions



netherphp / console build passing

✓ master rename the Print method. i KNEW i could not, but i did, and it worked. because it turns out in php 7 you can. but only php 7. not 5 nor hhvm.

Commit 504887f
Compare 533f955..504887f

Bob Majdak Jr authored and committed

#13 passed
Elapsed time 39 sec
Total time 1 min 53 sec
about a month ago

Build Jobs

✓	# 13.1	PHP: 5.4	no environment variables set	17 sec
✓	# 13.2	PHP: 5.5	no environment variables set	16 sec
✓	# 13.3	PHP: 5.6	no environment variables set	19 sec
✓	# 13.4	PHP: 7	no environment variables set	23 sec
✓	# 13.5	PHP: hhvm	no environment variables set	38 sec

Looser Restricted Word Restrictions

You can now use almost all keywords as method names, with the exception of "class".

Rejoyce, but also be careful.

public function private() { } // lol - now valid.

```php
class Whatever {
	public function
	Print() {
		echo __METHOD__, PHP_EOL;
	}

	public function
	ForEach() {
		echo __METHOD__, PHP_EOL;
	}

	public function
	List() {
	/*//
	this is the one that always used to piss me off the most
	when working on objects for storage and widgeting.
	//*/

		echo __METHOD__, PHP_EOL;
	}
}

$Whatever = new Whatever;

foreach(get_class_methods(get_class($Whatever)) as $Method)
$Whatever->{$Method}();
```

```
> php func\words1.php

Whatever::Print
Whatever::ForEach
Whatever::List
```

Cryptographic Safe RNG (random_bytes, random_int)

Generates random numbers using the proper cryptographically secure (as we know it) APIs on the various platforms.

- Windows: CryptGenRandom()
- Linux: getrandom()
- Fallback: /dev/urandom

```php
for($A = 0; $A < 5; $A++)
echo join(' ', str_split(bin2hex(
   random_bytes(10)
),2)), PHP_EOL;

for($A = 0; $A < 5; $A++)
echo random_int(PHP_INT_MIN, PHP_INT_MAX), PHP_EOL;
```

```
> php func\randombytes.php

6f 76 12 17 3b 5e b3 c9 6a 14
a8 a8 98 1d f8 2b 6e e6 2c ce
76 d4 d5 1d 10 91 ad 7d db ec
1e 9e a6 dc 51 31 6f 98 74 15
8d 73 4a 5e 50 4a ec a1 c6 e9

-5544930300178054325
-1189064797114373624
-6834803728182373946
5062669242465514688
7577228531150227048
```

Procedural Regex with preg_replace_callback_array

This function allows you to define a list of patterns to apply to the source data one at a time.

Nice for shorthanding.

Array of patterns where the key is the pattern and the value is the callback to execute. The return of the callback is what gets subbed in for the pattern.

```php
function Elitist($Found) {
    $Map = [
        'your'=>'ur', 'help'=>'halp',
        'a'=>'4', 'e'=>'3', 'o'=>'0', 's'=>'5', 't'=>'7',
        'h'=>'|-|', 'n'=>'|\|', 'm'=>'|\/|', 'p'=>'|*',
        'r'=>'|^', 'u'=>'|_|'
    ];

    return $Map[strtolower($Found)] ?? $Found;
}

$Original = 'Another settlement needs your help!';
$Altered = preg_replace_callback_array([
    '/\w+/'    => function($Match) { return Elitist($Match[0]); },
    '/[a-z]/i' => function($Match) { return Elitist($Match[0]); }
],$Original);

echo $Original, PHP_EOL;
echo $Altered, PHP_EOL;
```

```
> php func\pregreplacearr.php

Another settlement needs your help!

4|\|07|-|3|^ 53771З|\/|3|\|7 |\|33d5 |_||^ |-|41|*!
```

New Unicode Character Testing API (IntlChar)

Everything you ever needed to know about UTF characters and a lot of things you didn't want to know.

Kinda an odd API... all static...

With it though you can test anything about a character to determine how you should treat it.

```php
echo 'Codepoint for Train Is: ',
     dechex(IntlChar::CharFromName('STEAM LOCOMOTIVE')),
     PHP_EOL,
     PHP_EOL;

foreach(['@','1','a','!'] as $Char) {
    echo "Testing {$Char}", PHP_EOL;
    printf(
        'AlphaNum(%s) Digit(%s) Punctuation(%s)%s',
        (IntlChar::IsAlNum($Char))?('yes'):('no'),
        (IntlChar::IsDigit($Char))?('yes'):('no'),
        (IntlChar::IsPunct($Char))?('yes'):('no'),
        PHP_EOL
    );

    echo PHP_EOL;
}
```

```
> php other\intlchar.php
Codepoint for Train Is: 1f682

Testing @
AlphaNum(no) Digit(no) Punctuation(yes)

Testing 1
AlphaNum(yes) Digit(yes) Punctuation(no)

Testing a
AlphaNum(yes) Digit(no) Punctuation(no)

Testing !
AlphaNum(no) Digit(no) Punctuation(yes)
```

OLD THINGS

Stop using these things.

Deprecated Features

- PHP 4 style Constructors.
  - class Whatever { public function Whatever() { } }
- Calling static methods unstatically.
  - They have no $this.
  - Don't wait though on these, fix them now.
- Custom salt param on password_hash
  - Kinda a step backwards, but they think they are protecting you from yourself.
- Stream SSL Contexts: capture_session_meta
  - Session meta available via stream_get_meta_data

FIN

Bob Majdak Jr
@bobmagicii

Dallas PHP - March 2016

These Slides
https://goo.gl/wDX0eP

Code In Slides
https://goo.gl/EpCxCT

Documentation Sources

http://php.net/manual/en/migration70.php

http://php.net/manual/en/migration70.incompatible.php

http://php.net/manual/en/migration70.new-features.php

http://php.net/manual/en/migration70.new-functions.php

http://php.net/manual/en/migration70.classes.php

http://php.net/manual/en/migration70.changed-functions.php

http://php.net/manual/en/class.intlchar.php

http://php.net/manual/en/function.assert.php#function.assert.expectations