Secrets
ABAP
Apex
C
C++
CloudFormation
COBOL
C#
CSS
Flex
Go
HTML
Java
JavaScript
Kotlin
Objective C
PHP
PL/I
PL/SQL
Python
RPG
Ruby
Scala
Swift
Terraform
Text
TypeScript
T-SQL
VB.NET
VB6
XML

PHP static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your PHP code

All rules 268 | Vulnerability 40 | Bug 51 | Security Hotspot 33 | Code Smell 144

Tags

Search by name...

be used to end lines
Code Smell

More than one property should not be declared per statement
Code Smell

The "var" keyword should not be used
Code Smell

"<?php" and "<?=" tags should be used
Code Smell

File names should comply with a naming convention
Code Smell

Comments should not be located at the end of lines of code
Code Smell

Local variable and function parameter names should comply with a naming convention
Code Smell

Field names should comply with a naming convention
Code Smell

Lines should not end with trailing whitespaces
Code Smell

Files should contain an empty newline at the end
Code Smell

Modifiers should be declared in the correct order
Code Smell

An open curly brace should be located at the beginning of a line
Code Smell

Source code should comply with formatting standards

Analyze your code

Code Smell | Minor | convention psr2

Shared coding conventions make it possible for a team to collaborate efficiently. This rule raises issues for failures to comply with formatting standard. The default parameter values conform to the PSR2 standard.

Noncompliant Code Example

With the default PSR2 parameter values:

```
use FooClass;              // Noncompliant; the "use" declar

namespace Vendor\Package;
use FooClass;              // Noncompliant; the "namespace"
$foo = 1;                  // Noncompliant; the "use" declar

class ClassA {             // Noncompliant; an open curly br
  function my_function(){  // Noncompliant; curly brace on w
    if ($firstThing)       // Noncompliant; an open curly br
    {
      ...
    }

    if ($secondThing)    { // Noncompliant; there should be
      ...
    }

    if($thirdThing) {      // Noncompliant; there should be
      ...
    }
    else {                 // Noncompliant; the close curly
      ...
    }

    try{                   // Noncompliant; there should be
      ...
    } catch (Exception $e) {
    }

    analyse( $fruit ) ;    // Noncompliant; there should not

    for ($i = 0;$i < 10;   $i++) { // Nomcompliant; there sh
      ...
    }

    pressJuice($apply ,$orange);    // Noncompliant; the com

    do_something ();       // Noncompliant; there should not

    foreach ($fruits    as $fruit_key =>     $fruit) {  // N
      ...
    }
  }
}
```

An open curly brace should be located at the end of a line

Code Smell

Tabulation characters should not be used

Code Smell

Method and function names should comply with a naming convention

Code Smell

Creating cookies with broadly defined "domain" flags is security-sensitive

Security Hotspot

```
class ClassB
extends ParentClass  // Noncompliant; the class name and the
{
  ...
}

class ClassC extends ParentClass implements \ArrayAccess, \C
    \Serializable    // Noncompliant; the list of implemente
{

  public function aVeryLongMethodName(ClassTypeHint $arg1, /
    &$arg2, array $arg3 = []) {

    $noArgs_longVars = function () use ($longVar1,         /
        $longerVar2,
        $muchLongerVar3
    ) {
      ...
    };

    $foo->bar($longArgument,    // Noncompliant; the argumen
      $longerArgument,
      $muchLongerArgument);     // Noncompliant; the closing

    $closureWithArgsAndVars = function($arg1, $arg2)use   ($
      ...
    };
  }
}
```

Compliant Solution

```
namespace Vendor\Package; // Compliant; the "namespace" decl

use FooClass;             // Compliant; the "use" declaratio
                          // Compliant; the "use" declaratio
$foo = 1;

class ClassA
{                         // Compliant; the open curly brace
  function my_function()
  {                       // Compliant; the open curly brace
    if ($firstThing) {    // Compliant; the open curly brace
      ...
    }

    if ($secondThing) {   // Compliant; there is exactly one
      ...
    }

    if ($thirdThing) {    // Compliant; there is exactly one
      ...
    } else {              // Compliant; the close curly brac
      ...
    }

    try {                 // Compliant; there is exactly one
      ...
    } catch (Exception $e) {
      ...
    }

    analyse($fruit);      // Compliant: there is no space af

    for ($i = 0; $i < 10; $i++) { // Compliant: there is exa
      ...
    }

    pressJuice($apply, $orange);   // Compliant; the comma i

    do_something();       // Compliant; there is no space af

    foreach ($fruits as $fruit_key => $fruit) {  // Complian
      ...
    }
  }
}
```

```
/* The idea here is to make it obvious at first glance that
 * some other classes and/or implements some interfaces. The
 * extended classes or implemented interfaces can be located
 */
class ClassB1 extends ParentClass // Compliant; the class na
{
  ...
}

class ClassB2 extends             // Compliant; the class na
ParentClass {
  ...
}

/* Lists of implements may be split across multiple lines, w
 * is indented once. When doing so, the first item in the li
 * and there should be only one interface per line.
 */
class ClassC extends ParentClass implements
    \ArrayAccess,         // Compliant; the list of implemen
    \Countable,
    \Serializable
{
  /* Argument lists may be split across multiple lines, wher
   * is indented once. When doing so, the first item in the
   * and there should be only one argument per line. Also, w
   * split across multiple lines, the closing parenthesis an
   * placed together on their own line with one space betwee
   */
  public function aVeryLongMethodName(
    ClassTypeHint $arg1,  // Compliant; the arguments in a m
      &$arg2,
      array $arg3 = []
    ) {
      $noArgs_longVars = function () use (
        $longVar1,        // Compliant; the arguments in a m
        $longerVar2,
        $muchLongerVar3
      ) {
        ...
      };


    /* Argument lists may be split across multiple lines, wh
     * indented once. When doing so, the first item in the l
     * and there should be only one argument per line.
     */
    $foo->bar(
      $longArgument,       // Compliant; the arguments in th
      $longerArgument,
      $muchLongerArgument
    );                     // Compliant; the closing parenth

    /* Closures should be declared with a space after the "f
     * and a space before and after the "use" keyword.
     */
    $closureWithArgsAndVars = function ($arg1, $arg2) use ($
      ...
    };
  }
}
```

Available In:

sonarlint | sonarcloud | sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
Privacy Policy