Hi, thank you for your interest in contributing to the WordPress Coding Standards! We look forward to working with you.

# Reporting Bugs

Please search the repo to see if your issue has been reported already and if so, comment in that issue instead of opening a new one.

Before reporting a bug, you should check what sniff an error is coming from.

Running `phpcs` with the `-s` flag will show the name of the sniff with each error.

Bug reports containing a minimal code sample which can be used to reproduce the issue are highly appreciated as those are most easily actionable.

## Upstream Issues

Since WPCS employs many sniffs that are part of PHPCS, sometimes an issue will be caused by a bug in PHPCS and not in WPCS itself. If the error message in question doesn't come from a sniff whose name starts with `WordPress`, the issue is probably a bug in PHPCS itself, and should be [reported there](https://github.com/squizlabs/PHP\_CodeSniffer/issues).

# Contributing patches and new features

## Branches

Ongoing development will be done in the `develop` branch with merges done into `master` once considered stable.

To contribute an improvement to this project, fork the repo and open a pull request to the `develop` branch. Alternatively, if you have push access to this repo, create a feature branch prefixed by `feature/` and then open an intra-repo PR from that branch to `develop`.

Once a commit is made to `develop`, a PR should be opened from `develop` into `master` and named "Next release". This PR will provide collaborators with a forum to discuss the upcoming stable release.

# Considerations when writing sniffs

## Public properties

When writing sniffs, always remember that any `public` sniff property can be overruled via a custom ruleset by the end-user.

Only make a property `public` if that is the intended behaviour.

When you introduce new `public` sniff properties, or your sniff extends a class from which you inherit a `public` property, please don't forget to update the [public properties wiki page](https://github.com/WordPress/WordPress-Coding-Standards/wiki/Customizable-sniff-properties) with the relevant details once your PR has been merged into the `develop` branch.

# Unit Testing

## Pre-requisites

\* WordPress-Coding-Standards

\* PHP\_CodeSniffer 3.3.1 or higher

\* PHPUnit 4.x, 5.x, 6.x or 7.x

The WordPress Coding Standards use the `PHP\_CodeSniffer` native unit test suite for unit testing the sniffs.

Presuming you have installed `PHP\_CodeSniffer` and the WordPress-Coding-Standards as [noted in the README](https://github.com/WordPress/WordPress-Coding-Standards#how-to-use-this), all you need now is `PHPUnit`.

> N.B.: If you installed WPCS using Composer, make sure you used `--prefer-source` or run `composer install --prefer-source` now to make sure the unit tests are available.

> Other than that, you're all set already as Composer will have installed PHPUnit for you.

If you already have PHPUnit installed on your system: Congrats, you're all set.

## Installing PHPUnit

N.B.: \_If you used Composer to install the WordPress Coding Standards, you can skip this step.\_

You can either navigate to the directory where the `PHP\_CodeSniffer` repo is checked out and do `composer install` to install the `dev` dependencies or you can [install PHPUnit](https://phpunit.readthedocs.io/en/7.4/installation.html) as a PHAR file.

You may want to add the directory where PHPUnit is installed to a `PATH` environment variable for your operating system to make the command available everywhere on your system.

## Before running the unit tests

N.B.: \_If you used Composer to install the WordPress Coding Standards, you can skip this step.\_

For the unit tests to work, you need to make sure PHPUnit can find your `PHP\_CodeSniffer` install.

The easiest way to do this is to add a `phpunit.xml` file to the root of your WPCS installation and set a `PHPCS\_DIR` environment variable from within this file.

Copy the existing `phpunit.xml.dist` file and add the below `<env>` directive within the `<php>` section. Make sure to adjust the path to reflect your local setup.

```xml

<php>

<env name="PHPCS\_DIR" value="/path/to/PHP\_CodeSniffer/"/>

</php>

```

## Running the unit tests

\* If you didn't install WPCS using Composer, make sure you have registered the directory in which you installed WPCS with PHPCS using:

```sh

phpcs --config-set installed\_paths path/to/WPCS

```

\* Navigate to the directory in which you installed WPCS.

\* To run the unit tests:

```sh

phpunit --filter WordPress --bootstrap="/path/to/PHP\_CodeSniffer/tests/bootstrap.php" /path/to/PHP\_CodeSniffer/tests/AllTests.php

# Or if you've installed WPCS with Composer:

composer run-tests

```

Expected output:

```

PHPUnit 7.5.0 by Sebastian Bergmann and contributors.

Runtime: PHP 7.2.13

Configuration: /WordPressCS/phpunit.xml

........................................................ 56 / 56 (100%)

152 sniff test files generated 487 unique error codes; 52 were fixable (10.68%)

Time: 21.36 seconds, Memory: 22.00MB

OK (56 tests, 0 assertions)

```

[![asciicast](https://asciinema.org/a/98078.png)](https://asciinema.org/a/98078)

## Unit Testing conventions

If you look inside the `WordPress/Tests` subdirectory, you'll see the structure mimics the `WordPress/Sniffs` subdirectory structure. For example, the `WordPress/Sniffs/PHP/POSIXFunctionsSniff.php` sniff has its unit test class defined in `WordPress/Tests/PHP/POSIXFunctionsUnitTest.php` which checks the `WordPress/Tests/PHP/POSIXFunctionsUnitTest.inc` test case file. See the file naming convention?

Lets take a look at what's inside `POSIXFunctionsUnitTest.php`:

```php

...

namespace WordPressCS\WordPress\Tests\PHP;

use PHP\_CodeSniffer\Tests\Standards\AbstractSniffUnitTest;

class POSIXFunctionsUnitTest extends AbstractSniffUnitTest {

/\*\*

\* Returns the lines where errors should occur.

\*

\* @return array <int line number> => <int number of errors>

\*/

public function getErrorList() {

return array(

13 => 1,

16 => 1,

18 => 1,

20 => 1,

22 => 1,

24 => 1,

26 => 1,

);

}

...

```

Also note the class name convention. The method `getErrorList()` MUST return an array of line numbers indicating errors (when running `phpcs`) found in `WordPress/Tests/PHP/POSIXFunctionsUnitTest.inc`.

If you run:

```sh

$ cd /path-to-cloned/phpcs

$ ./bin/phpcs --standard=Wordpress -s /path/to/WordPress/Tests/PHP/POSIXFunctionsUnitTest.inc --sniffs=WordPress.PHP.POSIXFunctions

...

--------------------------------------------------------------------------------

FOUND 7 ERRORS AFFECTING 7 LINES

--------------------------------------------------------------------------------

13 | ERROR | ereg() has been deprecated since PHP 5.3 and removed in PHP 7.0,

| | please use preg\_match() instead.

| | (WordPress.PHP.POSIXFunctions.ereg\_ereg)

16 | ERROR | eregi() has been deprecated since PHP 5.3 and removed in PHP 7.0,

| | please use preg\_match() instead.

| | (WordPress.PHP.POSIXFunctions.ereg\_eregi)

18 | ERROR | ereg\_replace() has been deprecated since PHP 5.3 and removed in PHP

| | 7.0, please use preg\_replace() instead.

| | (WordPress.PHP.POSIXFunctions.ereg\_replace\_ereg\_replace)

20 | ERROR | eregi\_replace() has been deprecated since PHP 5.3 and removed in PHP

| | 7.0, please use preg\_replace() instead.

| | (WordPress.PHP.POSIXFunctions.ereg\_replace\_eregi\_replace)

22 | ERROR | split() has been deprecated since PHP 5.3 and removed in PHP 7.0,

| | please use explode(), str\_split() or preg\_split() instead.

| | (WordPress.PHP.POSIXFunctions.split\_split)

24 | ERROR | spliti() has been deprecated since PHP 5.3 and removed in PHP 7.0,

| | please use explode(), str\_split() or preg\_split() instead.

| | (WordPress.PHP.POSIXFunctions.split\_spliti)

26 | ERROR | sql\_regcase() has been deprecated since PHP 5.3 and removed in PHP

| | 7.0, please use preg\_match() instead.

| | (WordPress.PHP.POSIXFunctions.ereg\_sql\_regcase)

--------------------------------------------------------------------------------

....

```

You'll see the line number and number of ERRORs we need to return in the `getErrorList()` method.

The `--sniffs=...` directive limits the output to the sniff you are testing.

## Code Standards for this project

The sniffs and test files - not test \_case\_ files! - for WPCS should be written such that they pass the `WordPress-Extra` and the `WordPress-Docs` code standards using the custom ruleset as found in `/.phpcs.xml.dist`.