PHP Coding standards

Basics

KaunasPHP v.29 - CodeWeek Edition

Linas Gricius

- PHP developer over 12 years
- Zend Certified PHP Engineer
- System Architect at **Intermedix Lietuva**
- Before: senior developer / team lead at Lamoda.ru

PHP Framework Interop Group - www.php-fig.org

- PSR stands for PHP Standard Recommendation

- PSR-1 - Basic Coding Standard
- PSR-2 - Coding Style Guide
- PSR-4 - Autoloading Standard

- PSR-5 - PHPDoc (Draft!)

PSR-1 - Basic Coding Standard

- Files MUST use only <?php and <?= tags.
- Files MUST use only UTF-8 without BOM for PHP code.
- Files SHOULD either declare symbols (classes, functions, constants, etc.) or cause side-effects (e.g. generate output, change .ini settings, etc.) but SHOULD NOT do both.
- Namespaces and classes MUST follow an "autoloading" PSR-4 (PSR0 is deprecated!)
- Class names MUST be declared in StudlyCaps.
- Class constants MUST be declared in all uppercase with underscore separators.
- Method names MUST be declared in camelCase

PSR-2 - Coding Style Guide (1)

- Code MUST follow a "coding style guide" PSR-1.
- Code MUST use 4 spaces for indentation, not tabs.
- There MUST NOT be a hard limit on line length; the soft limit MUST be 120 characters; lines SHOULD be 80 characters or less.
- There MUST be one blank line after the namespace declaration, and there MUST be one blank line after the block of use declarations.
- Opening braces for classes MUST go on the next line, and closing braces MUST go on the next line after the body.
- Opening braces for methods MUST go on the next line, and closing braces MUST go on the next line after the body.

PSR-2 - Coding Style Guide (2)

- Visibility MUST be declared on all properties and methods; abstract and final MUST be declared before the visibility; static MUST be declared after the visibility.
- Control structure keywords MUST have one space after them; method and function calls MUST NOT.
- Opening braces for control structures MUST go on the same line, and closing braces MUST go on the next line after the body.
- Opening parentheses for control structures MUST NOT have a space after them, and closing parentheses for control structures MUST NOT have a space before.

PSR-4 - Autoloader

- As of 2014-10-21 PSR-0 has been marked as deprecated. PSR-4 is now recommended as an alternative.

- This PSR describes a specification for autoloading classes from file paths. It is fully interoperable, and can be used in addition to any other autoloading specification, including PSR-0. This PSR also describes where to place files that will be autoloaded according to the specification.

PHPDoc and PSR-5 (draft)

- phpDocumentor is the world standard auto-documentation tool for PHP. Written in PHP, phpDocumentor can be used directly from the command-line. phpDocumentor can be used to generate professional documentation directly from the source code of your PHP project.

- The main purpose of this PSR-5 is to provide a complete and formal definition of the PHPDoc standard. This PSR deviates from its predecessor, the de-facto PHPDoc Standard associated with phpDocumentor 1.x, to provide support for newer features in the PHP language and to address some of the shortcomings of its predecessor.

PHP Mess Detector (PHPMD)

It takes a given PHP source code base and look for several potential problems within that source.

These problems can be things like:

- Possible bugs
- Suboptimal code
- Overcomplicated expressions
- Unused parameters, methods, properties

PHP_CodeSniffer (PHPCS)

PHP_CodeSniffer tokenizes PHP, JavaScript and CSS files and detects violations of a defined set of coding standards.

JSLint / JSHint

- JSLint is a JavaScript program that looks for problems in JavaScript programs. It is a code quality tool.

- JSHint, A Static Code Analysis Tool for JavaScript

Customizing Git - Git Hooks

Like many other Version Control Systems, Git has a way to fire off custom scripts when certain important actions occur. There are two groups of these hooks: client-side and server-side. Client-side hooks are triggered by operations such as committing and merging, while server-side hooks run on network operations such as receiving pushed commits. You can use these hooks for all sorts of reasons.

Setting up git hook

```
➜  php-coding-standards git:(master) ✗ cd .git/hooks

➜  hooks git:(master) ln -s ../../.githooks/pre-commit pre-commit

➜  hooks git:(master) chmod +x pre-commit

➜  hooks git:(master) cat ../../.git/hooks/pre-commit

#!/bin/sh

➜  hooks git:(master) cd ../../
```

pre-commit (1)

```
#!/bin/sh

HAS_ERRORS=0

PROJECT=`php -r "echo dirname(dirname(realpath('$0')));"`

STAGED_FILES_CMD=`git diff --cached --name-only | grep \\.php$`



SFILES=${SFILES:-$STAGED_FILES_CMD}
```

pre-commit (2)

```
echo "Checking PHP Lint...\n"
for FILE in $SFILES
do
    php -l $PROJECT/$FILE
    if [ $? != 0 ]
    then
        HAS_ERRORS=1;
    fi
done
```

pre-commit (3)

```
echo "Running Code Sniffer...\n"
for FILE in $SFILES do
    vendor/bin/phpcs --standard=PSR2 $PROJECT/$FILE -n
    if [ $? != 0 ]
    then
        HAS_ERRORS=1;
    fi
done
```

pre-commit (4)

```
echo "Running Mess detector...\n"
for FILE in $SFILES do
    vendor/bin/phpmd $PROJECT/$FILE text cleancode,codesize,unusedcode
  if [ $? != 0 ]
  then
    HAS_ERRORS=1;
  fi
done
```

pre-commit (5)

```
if [ $HAS_ERRORS -gt 0 ]
then
    echo "\n\n"
    echo "Fix errors/warnings before commit!"
    exit 1;
fi
exit $?
```

Links

- http://www.php-fig.org/faq/
- http://www.phpdoc.org/
- http://phpmd.org/
- https://github.com/squizlabs/PHP_CodeSniffer
- http://www.jslint.com/help.html
- http://www.jslint.com/help.html
- https://github.com/phpDocumentor/fig-standards/blob/master/proposed/phpdoc.md
- https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks
- https://gist.github.com/ronanguilloux/11f6a788358577474ab4

This is it.