Cool Tools for PHP Development

Ben Ramsey

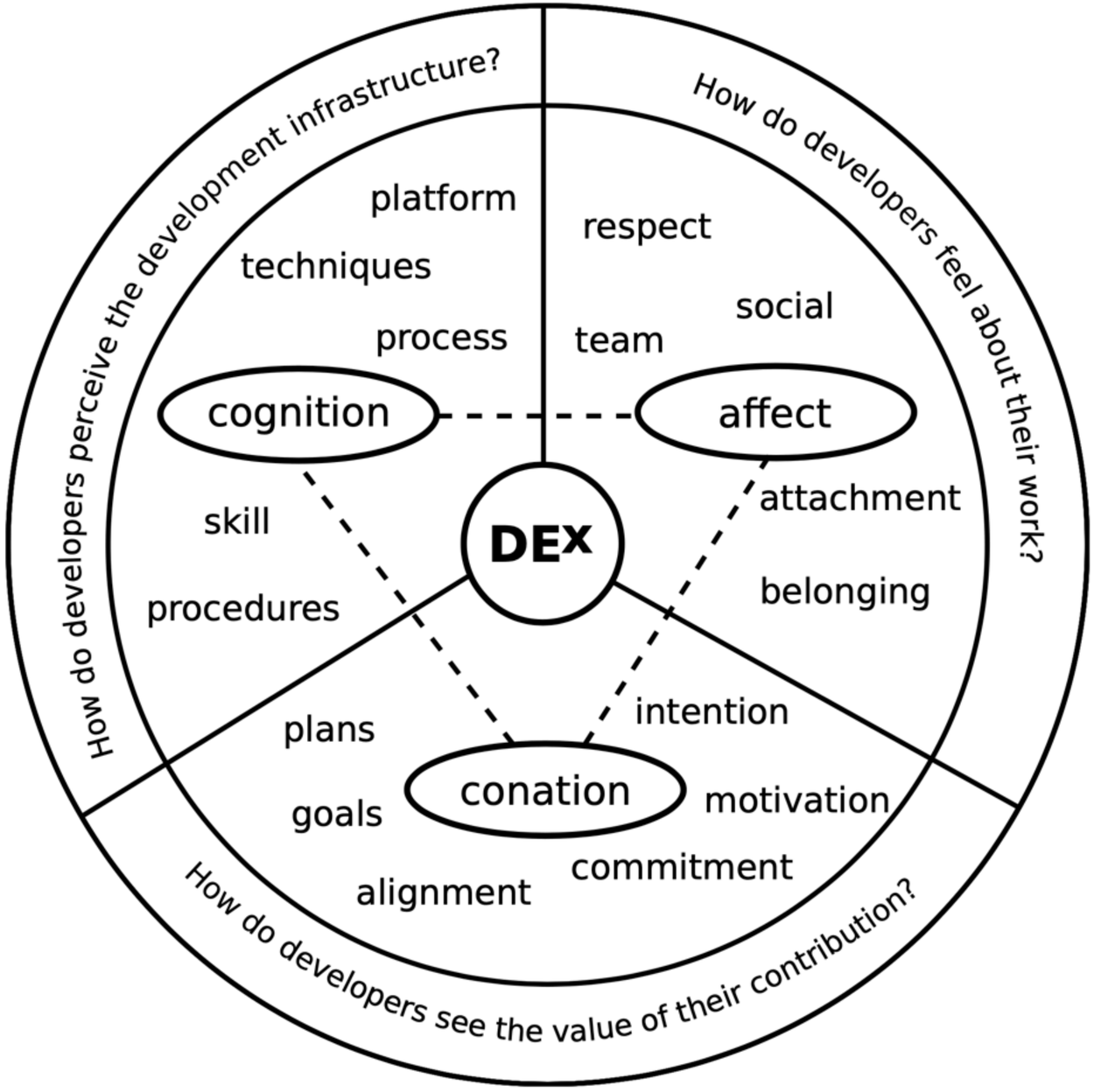
php[tek] • 17 May 2023

What is DX?

DEVELOPER EXPERIENCE

"DE^{x} consists of experiences relating to all kinds of artifacts and activities that a developer may encounter as part of their involvement in software development. These could roughly be divided into experiences regarding i) development infrastructure (e.g. development and management tools, programming languages, libraries, platforms, frameworks, processes, and methods), ii) feelings about work (e.g. respect, attachment, belonging), and iii) the value of one's own contribution (e.g. alignment of one's own goals with those of the project, plans, intentions, and commitment)."

Fagerholm, F., Münch, J. (2012). Developer Experience: Concept and Definition. In International Conference on Software and System Process (ICSSP 2012), pp. 73–77. IEEE.



Fagerholm, F., Münch, J. (2012). Developer Experience: Concept and Definition. In International Conference on Software and System Process (ICSSP 2012), pp. 73–77. IEEE.

Developer experience

CONCEPTUAL FRAMEWORK

Cognition

How do developers perceive the development infrastructure?

Conation

How do developers see the value of their contribution?

Affect

How do developers feel about their work?

Fagerholm, F., Münch, J. (2012). Developer Experience: Concept and Definition. In International Conference on Software and System Process (ICSSP 2012), pp. 73–77. IEEE.

How do you perceive the infrastructure?

Do your tools get in the way?

Are they a joy to use?

The better experience a developer has with their tools, the more efficient and happy they'll be.


Lastly, ask yourself, does it spark joy? Does it make you happy? Only keep those items that do!

Cool tools

Cool tools

WHAT ARE THEY?

The tools you need to do your job

The tools that make your job easier

The tools that bring joy to your work

Cool tools

OUR FOCUS

How can you create better workflows for the tools you already use?

Programming is a “team sport.” How can you share these workflows with your team?

In other words, how can you improve your DX?

Starting Out

Unit tests?

Of course we want those!

So, let's install PHPUnit.

```
composer require --dev phpunit/phpunit
```

Coding standards?

Yes, please!

Let’s install PHP_CodeSniffer.

```
composer require --dev \
    squizlabs/php_codesniffer
```

Static analysis?

You bet!

Let’s install PHPStan.

And, for the heck of it, Psalm, too!

```
composer require --dev \
    phpstan/phpstan \
    vimeo/psalm
```

Now what?

Now, we write our code...

And then run the commands for our tools.

src/Example.php

```php
namespace Ramsey\CoolTools;

class Example
{
    public function greet(string $name = 'World'): string
    {
        return "Hello, {$name}!";
    }
}
```

tests/ExampleTest.php

```php
namespace Ramsey\Test\CoolTools;

use PHPUnit\Framework\TestCase;
use Ramsey\CoolTools\Example;

class ExampleTest extends TestCase
{
    public function testGreet(): void
    {
        $example = new Example();
        $this->assertSame(
            'Hello, Friends!', $example->greet('Friends')
        );
    }
}
```

```
./vendor/bin/phpunit
```

```
PHPUnit 10.1.2 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.4
Configuration: /home/ramsey/cool-tools/phpunit.xml.dist

.                                                                   1 / 1 (100%)

Time: 00:00.008, Memory: 8.00 MB

OK (1 test, 1 assertion)
```

```
./vendor/bin/phpcs
```

```
.. 2 / 2 (100%)

Time: 46ms; Memory: 8MB
```

```
./vendor/bin/phpstan
```

```
Note: Using configuration file /home/ramsey/cool-tools/phpstan.neon.dist.
 2/2 [▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓] 100%

 [OK] No errors
```

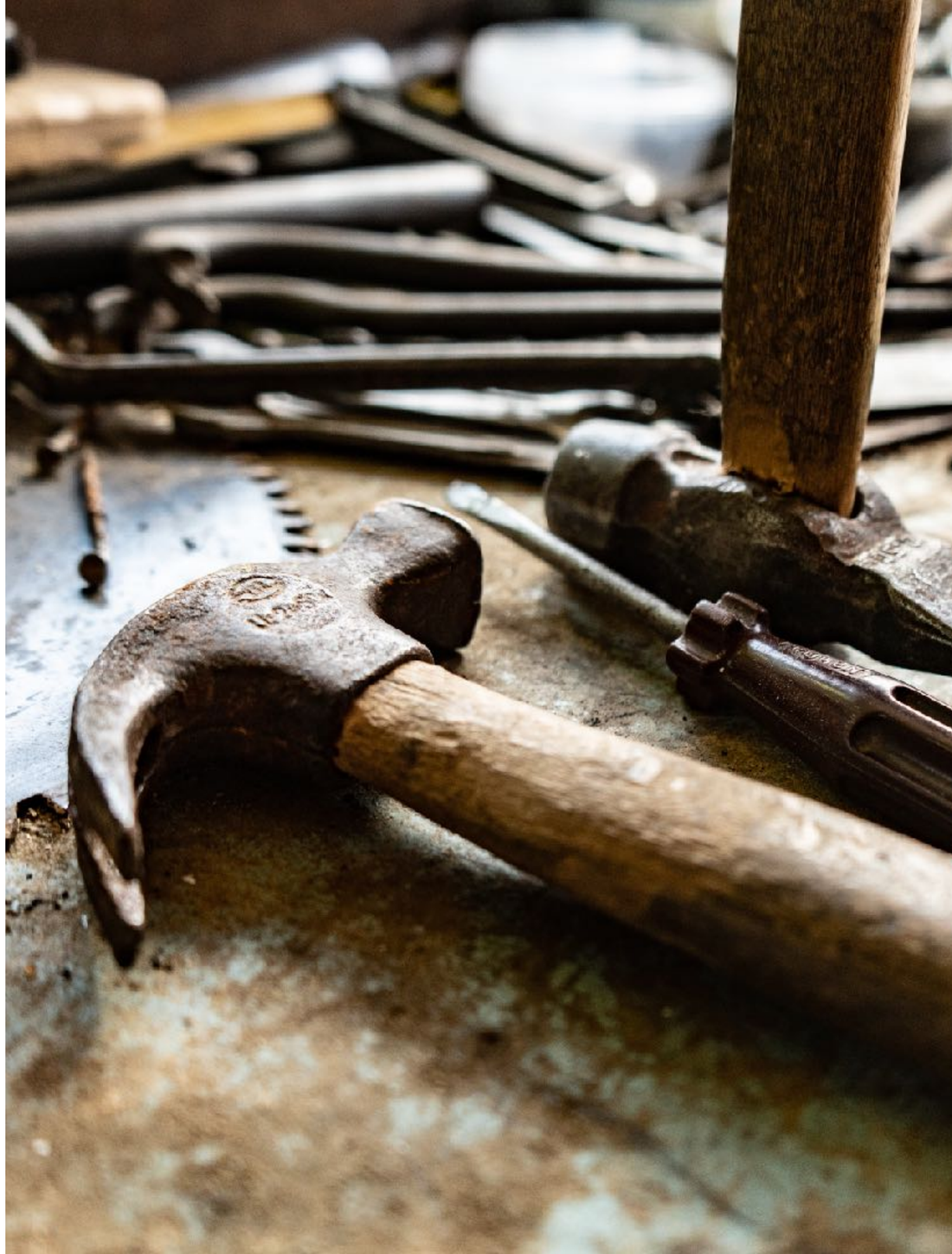
```
./vendor/bin/psalm
```

```
Target PHP version: 8.2 (inferred from current PHP version).
Scanning files...
Analyzing files...

▒
------------------------------
No errors found!
------------------------------

Checks took 1.12 seconds and used 172.273MB of memory
Psalm was able to infer types for 100% of the codebase
```

Leveling Up



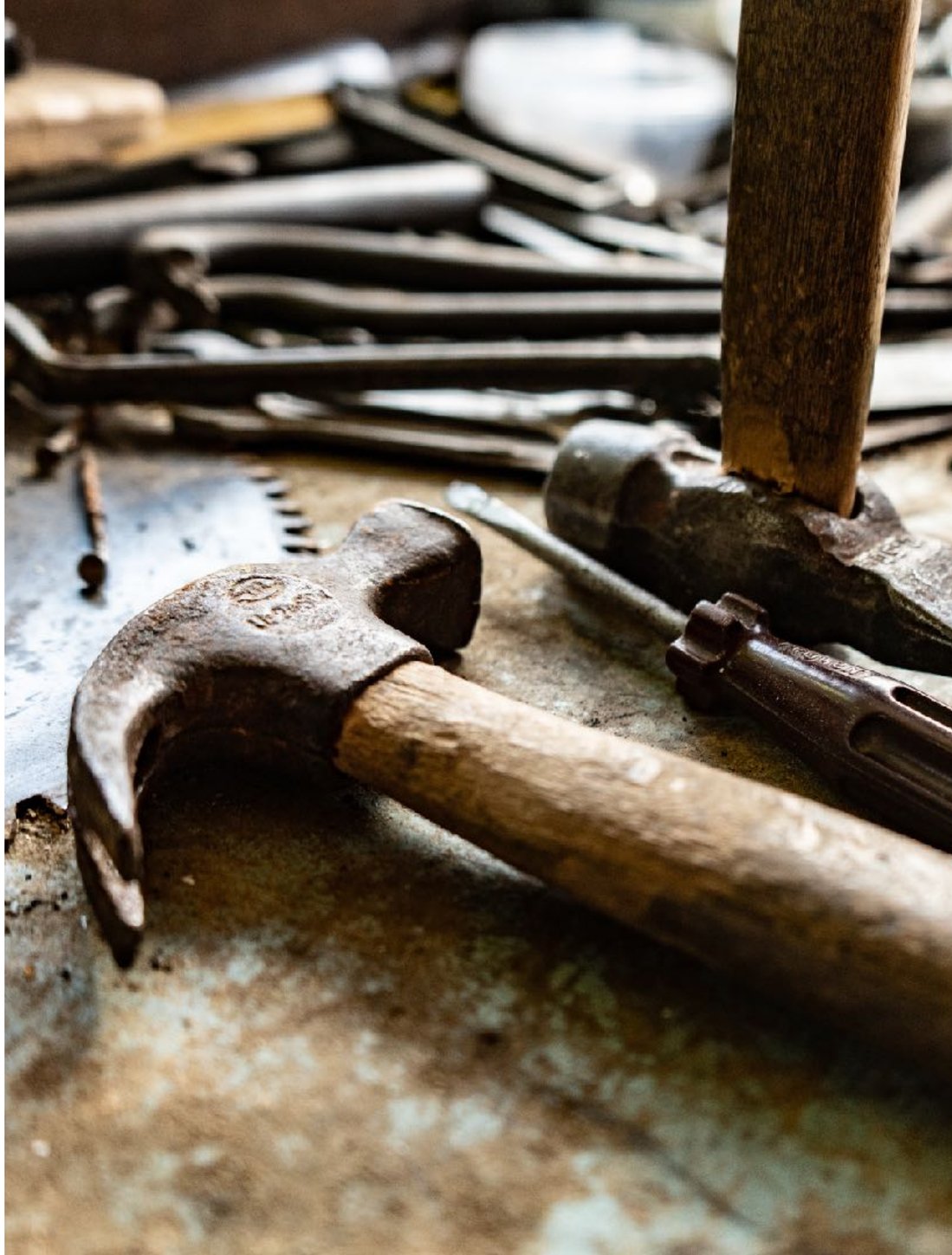
Composer scripts

PHP callbacks (static methods)

Command-line executable commands

Symfony Console Command classes

Event hooks



composer.json

```
{
  "scripts": {
    "test": ["phpcs", "phpstan", "psalm", "phpunit"]
  }
}
```

```
composer test
```

composer.json

```
{
  "scripts": {
    "dev:analyze:phpstan": "phpstan analyse --memory-limit=1G",
    "dev:analyze:psalm": "psalm"
  }
}
```

composer.json

```
{
  "scripts": {
    "dev:analyze": [
      "@dev:analyze:phpstan",
      "@dev:analyze:psalm"
    ],
    "dev:analyze:phpstan": "phpstan analyse --memory-limit=1G",
    "dev:analyze:psalm": "psalm"
  }
}
```

```
composer list dev
```

```
Available commands for the "dev" namespace:
  dev:analyze              Runs the dev:analyze script as defined in composer.json
  dev:analyze:phpstan      Runs the dev:analyze:phpstan script as defined in composer.json
  dev:analyze:psalm        Runs the dev:analyze:psalm script as defined in composer.json
```

composer.json

```
{
  "scripts-descriptions": {
    "dev:analyze": "Runs all static analysis checks.",
    "dev:analyze:phpstan": "Runs the PHPStan static analyzer.",
    "dev:analyze:psalm": "Runs the Psalm static analyzer."
  }
}
```

```
Available commands for the "dev" namespace:
  dev:analyze              Runs all static analysis checks.
  dev:analyze:phpstan      Runs the PHPStan static analyzer.
  dev:analyze:psalm        Runs the Psalm static analyzer.
```

composer.json

```
{
  "scripts": {
    "dev:analyze": [
      "@dev:analyze:phpstan",
      "@dev:analyze:psalm"
    ],
    "dev:analyze:phpstan": "phpstan analyse --ansi --memory-limit=1G",
    "dev:analyze:psalm": "psalm",
    "dev:build:clean": "git clean -fX build/",
    "dev:lint": [
      “@dev:lint:syntax",
      "@dev:lint:style"
    ],
    "dev:lint:fix": "phpcbf",
    "dev:lint:style": "phpcs --colors",
    "dev:lint:syntax": "parallel-lint --colors src/ tests/",
    "dev:test": [
      "@dev:lint",
      "@dev:analyze",
      "@dev:test:unit"
    ],
    "dev:test:coverage:ci": "phpunit --colors=always --coverage-text --coverage-clover build/coverage/clover.xml --coverage-
cobertura build/coverage/cobertura.xml --coverage-crap4j build/coverage/crap4j.xml --coverage-xml build/coverage/coverage-xml --
log-junit build/junit.xml",
    "dev:test:coverage:html": "phpunit --colors=always --coverage-html build/coverage/coverage-html/",
    "dev:test:unit": "phpunit --colors=always",
    "test": "@dev:test"
  }
}
```

Sharing Tools

Composer plugins

Extend the functionality provided by Composer

Hook into Composer events

Add commands to Composer

Use the same plugin across projects

Plugin examples

ramsey/composer-repl

ramsey/devtools

ramsey/composer-repl

Uses PsySH to provide a REPL that can interact with your project

Composer plugin with a single command: `composer repl`

Good example of minimal Composer plugin

composer.json (ramsey/composer-repl)

```
{
    "name": "ramsey/composer-repl",
    "type": "composer-plugin",
    "require": {
        "composer-plugin-api": "^2.3"
    },
    "require-dev": {
        "composer/composer": "^2.3"
    },
    "extra": {
        "class": "Ramsey\\Dev\\Repl\\Composer\\ReplPlugin"
    }
}
```

src/Composer/ReplPlugin.php (ramsey/composer-repl)

```php
use Composer\Plugin\Capability\CommandProvider;
use Composer\Plugin\Capable;
use Composer\Plugin\PluginInterface;

class ReplPlugin implements Capable, CommandProvider, PluginInterface
{
    public function getCapabilities(): array
    {
        return [CommandProvider::class => self::class];
    }

    public function getCommands(): array
    {
        return [new ReplCommand()];
    }

    public function activate(Composer $composer, IOInterface $io): void {}

    public function deactivate(Composer $composer, IOInterface $io): void {}

    public function uninstall(Composer $composer, IOInterface $io): void {}
}
```

src/Composer/ReplCommand.php (ramsey/composer-repl)

```php
use Composer\Command\BaseCommand;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;

class ReplCommand extends BaseCommand
{
  protected function configure(): void
  {
    $this->setName('repl')
      ->setDescription('Launches a development console (REPL) for PHP.')
      ->setAliases(['shell']);
  }

  protected function execute(InputInterface $input, OutputInterface $output): int
  {
    Config::disableProcessTimeout();

    return runReplCommand();
  }
}
```

```
〉composer require --dev ramsey/composer-repl

...

〉composer list

...

Available commands:
  ...
  repl                  [shell] Launches a development console (REPL) for PHP.
  ...


〉composer repl
```

ramsey/devtools

Provides a lot of common functionality I use across my open source projects

Composer plugin with many commands

Good example of complex Composer plugin

Extensible, so you can build off of it

composer.json (ramsey/devtools)

```
{
    "name": "ramsey/composer-repl",
    "type": "composer-plugin",
    "require": {
        "composer-plugin-api": "^2.3"
    },
    "require-dev": {
        "composer/composer": "^2.3"
    },
    "extra": {
        "class": "Ramsey\\Dev\\Tools\\Composer\\DevToolsPlugin"
    }
}
```

There's some hand-waving here...

ramsey/devtools is really two packages:

- ramsey/devtools
- ramsey/devtools-lib

Since we cannot extend Composer plugins; devtools-lib allows easier extension

composer.json

```
{
  "scripts": {
    "dev:analyze": [
      "@dev:analyze:phpstan",
      "@dev:analyze:psalm"
    ],
    "dev:analyze:phpstan": "phpstan analyse --ansi --memory-limit=1G",
    "dev:analyze:psalm": "psalm",
    "dev:build:clean": "git clean -fX build/",
    "dev:lint": [
      “@dev:lint:syntax",
      "@dev:lint:style"
    ],
    "dev:lint:fix": "phpcbf",
    "dev:lint:style": "phpcs --colors",
    "dev:lint:syntax": "parallel-lint --colors src/ tests/",
    "dev:test": [
      "@dev:lint",
      "@dev:analyze",
      "@dev:test:unit"
    ],
    "dev:test:coverage:ci": "phpunit --colors=always --coverage-text --coverage-clover build/coverage/clover.xml --coverage-
cobertura build/coverage/cobertura.xml --coverage-crap4j build/coverage/crap4j.xml --coverage-xml build/coverage/coverage-xml --
log-junit build/junit.xml",
    "dev:test:coverage:html": "phpunit --colors=always --coverage-html build/coverage/coverage-html/",
    "dev:test:unit": "phpunit --colors=always",
    "test": "@dev:test"
  }
}
```

Let’s look at some code...

DevToolsApplication

DevToolsPlugin

UnitCommand

Enforcing Workflows

Avast!

SAY “HELLO” TO THE CAPTAIN

CaptainHook

Git hook manager

Attach workflows to Git hooks to ensure everyone uses the same workflows

Extend it with your own hooks

captainhook.json

```
{
  "commit-msg": {
    "enabled": true,
    "actions": []
  },
  "pre-push": {
    "enabled": true,
    "actions": []
  },
  "pre-commit": {
    "enabled": true,
    "actions": []
  },
  "prepare-commit-msg": {
    "enabled": true,
    "actions": []
  },
  "post-commit": {
    "enabled": false,
    "actions": []
  },
  "post-merge": {
    "enabled": true,
    "actions": []
  },
  "post-checkout": {
    "enabled": true,
    "actions": []
  },
  "post-rewrite": {
    "enabled": false,
    "actions": []
  },
  "post-change": {
    "enabled": false,
    "actions": []
  }
}
```

captainhook.json

```
{
  "pre-commit": {
    "enabled": true,
    "actions": [{
      "action": "composer validate",
      "conditions": [{
        "exec": "\\CaptainHook\\App\\Hook\\Condition\\FileStaged\\Any",
        "args": [["composer.json"]]
      }]
    }, {
      "action": "composer normalize --dry-run",
      "conditions": [{
        "exec": "\\CaptainHook\\App\\Hook\\Condition\\FileStaged\\Any",
        "args": [["composer.json"]]
      }]
    }, {
      "action": "composer dev:lint:syntax -- {$STAGED_FILES|of-type:php}",
      "conditions": [{
        "exec": "\\CaptainHook\\App\\Hook\\Condition\\FileStaged\\OfType",
        "args": ["php"]
      }]
    }, {
      "action": "composer dev:lint:style -- {$STAGED_FILES|of-type:php}",
      "conditions": [{
        "exec": "\\CaptainHook\\App\\Hook\\Condition\\FileStaged\\OfType",
        "args": ["php"]
      }]
    }]
  }
}
```

captainhook.json

```
{
  "prepare-commit-msg": {
    "enabled": true,
    "actions": [{
      "action": "\\Ramsey\\CaptainHook\\PrepareConventionalCommit"
    }]
  },
  "commit-msg": {
    "enabled": true,
    "actions": [{
      "action": "\\Ramsey\\CaptainHook\\ValidateConventionalCommit"
    }]
  }
}
```

captainhook.json

```
{
  "pre-push": {
    "enabled": true,
    "actions": [
      {
        "action": "composer test"
      }
    ]
  }
}
```

captainhook.json

```
{
  "post-merge": {
    "enabled": true,
    "actions": [{
      "action": "composer install --ansi",
      "conditions": [{
        "exec": "\\CaptainHook\\App\\Hook\\Condition\\FileChanged\\Any",
        "args": [["composer.json", "composer.lock"]]
      }]
    }]
  },
  "post-checkout": {
    "enabled": true,
    "actions": [{
      "action": "composer install --ansi",
      "conditions": [{
        "exec": "\\CaptainHook\\App\\Hook\\Condition\\FileChanged\\Any",
        "args": [["composer.json", "composer.lock"]]
      }]
    }]
  }
}
```

composer.json - Configuring CaptainHook

```
{
  "require-dev": {
    "captainhook/plugin-composer": "^5.3",
  },
  "config": {
    "allow-plugins": {
      "captainhook/plugin-composer": true
    }
  },
  "extra": {
    "captainhook": {
      "force-install": true
    }
  }
}
```

Hooray!

Recap

WE COVERED...

Using Composer scripts to stitch together workflows and share with team/contributors

Using Composer plugins to combine workflows into commands that can be shared across projects

Using CaptainHook to ensure team/ contributors follow workflows seamlessly

Learn more

Composer scripts:
getcomposer.org/doc/articles/scripts.md

Composer plugins:
getcomposer.org/doc/articles/plugins.md

CaptainHook:
captainhookphp.github.io/captainhook

Lots of cool tools!

atoum

Behat

bitexpert/captainhook-infection

CaptainHook

Code Scrawl

Codeception

composer outdated -D

composer validate

composer-lock-diff

composer-normalize

composer-unused

ComposerRequireChecker

Danger PHP

dePHPend

Deptrac

doctrine orm:validate-schema

Dredd

Exakat

GrumPHP

Infection

laminas/automatic-releases

Lando

Latte syntax checker

madewithlove/license-checker

mamazu/documentation-validator

Markdown Link Linter

mlocati/docker-php-extension-installer

Neon checker

Parse: A PHP Security Scanner

Pest

Phan

Phinx: Simple PHP Database Migrations

PHIVE

php -l

PHP Architecture Tester

PHP Copy/Paste Detector

PHP CS Fixer

PHP Extensions Finder

PHP Insights

PHP Magic Number Detector

PHP Mess Detector

PHP Parallel Lint

PHP Quality Assurance

PHP Security Advisories Database

PHP VarDump Check

PHPBench

PHPCompatibility

PHPCSExtra

phpDocumentor
phpDox
PHPStan
Phpsu: Synchronisation Utility
PHPUnit
PHP_CodeSniffer
Prettier
Psalm
ramsey/composer-install
ramsey/composer-repl
ramsey/conventional-commits
ramsey/devtools
Rector
Roave Backward Compatibility Check
Roave Security Advisories
roave/no-floaters
roave/no-leaks
roave/you-are-using-it-wrong
Robo
Slevomat Coding Standard
symfony check:security
symplify/easy-coding-standard
UpToDocs

And many more at packagist.org

Thanks!

- joind.in/talk/21a01
- phpc.social/@ramsey
- github.com/ramsey
- ben@benramsey.com