FOOMAN

Magento 2 Quick Tips

10 QUICK, RANDOM WAYS TO EASIER MAGENTO 2 DEVELOPMENT

CONTENTS

10 Random Ways to Easier Magento 2 Development

Keeping Magento 2 Admin Settings Organised

There are a few things that come included in Magento 2 which help in keeping your admin settings easy to use.

Hide Unused Settings

First - it's always a good idea to remove as many settings as possible.
This obviously needs to be weighed up with the need to make your module customisable. One thing which helps is to hide settings which don't have any impact with the current configuration. In other words, if a certain setting only makes sense when another specific setting is set to a particular value, then we can hide it. Magento offers this out of the box using the <depends> node.

```
<!-- etc/adminhtml/system.xml-->
<field id="exportwithstatus" translate="label" sortOrder="61" type="multiselect"
showInDefault="1"
       showInWebsite="1" showInStore="1">
    <label>Export Orders with Status</label>
    <source_model>Fooman\Connect\Model\System\OrderStatusOptions</source_model>
    <depends>
        <field id="exportmode">order</field>
    </depends>
```

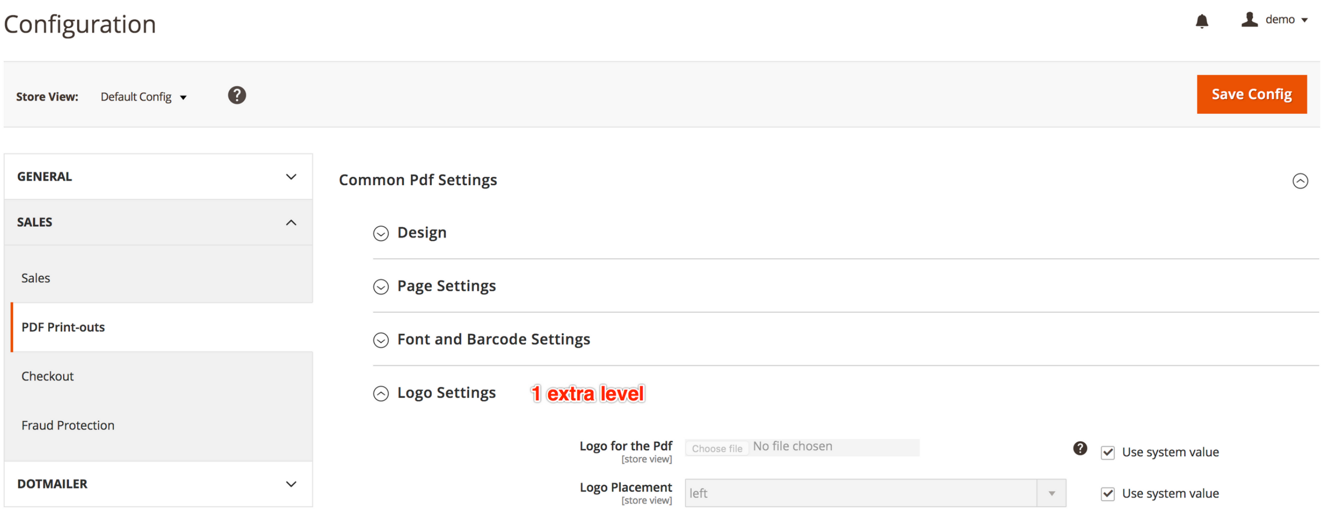
You can even do something like:

```
<!-- etc/adminhtml/system.xml-->
<field id="paypalrefundbankaccount" translate="label" sortOrder="425" type="select"
showInDefault="1"
       showInWebsite="1" showInStore="1">
    <label>Bank Account for Paypal Refunds</label>
    <source_model>Fooman\Connect\Model\System\BankAccountOptions</source_model>
    <depends>
        <field id="xerostatus">AUTHORISED</field>
        <field id="cashrefund">1</field>
    </depends>
```

This will only show the above setting if the two other settings (xerostatus and cashrefund) have been set to a certain value:

Group Similar Settings

With Magento 2 you can add one extra layer to group similar settings together, compared to Magento 1:



Logo for the Pdf [store view] — Choose file — No file chosen — ✓ Use system value

Logo Placement [store view] — left — ✓ Use system value

To use it, nest an additional level of <group> nodes and Magento will take care of the open/shut toggles:

```
<!-- etc/adminhtml/system.xml-->
<system>
    <section id="sales_pdf">
        <group id="all" translate="label" type="text" sortOrder="10" showInDefault="1"
showInWebsite="1"
               showInStore="1">
            <label>Common Pdf Settings</label>
            <group id="page" translate="label" type="text" sortOrder="100"
showInDefault="1" showInWebsite="1"
                   showInStore="1" canRestore="1">
                <label>Page Settings</label>
                <field id="allpagesize" translate="label" type="select" sortOrder="100"
showInDefault="1"
                       showInWebsite="0" showInStore="0">
                    <label>Page Size</label>
```

Migrating Settings

One trick to keep in mind is - as your module develops, your settings likely evolve over time as well. This could lead to the need to re-organise your settings and shift them around to a new group to keep them easy to use. Without any further action on your part this could unfortunately mean that your existing users would either need to reconfigure the extension or you would need to write some upgrade data script that migrates your settings to the new path. Fortunately there is a third option which makes this super easy:

```
<!-- etc/adminhtml/system.xml-->
<field id="displayboth" translate="label" type="select" sortOrder="600"
showInDefault="1"
       showInWebsite="1" showInStore="1" canRestore="1">
    <label>Display Base and Order Currency</label>
    <source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
    <config_path>sales_pdf/all/displayboth</config_path>
```

By adding a config_path node with the old path, we have the best of both worlds - an easier to work with admin area, while still keeping the original settings intact.

Magento 2 / Php Version Matrix

A quick list containing the first and latest version of the main Magento 2 branches and their respective supported PHP versions:

Magento Version	Supported PHP Versions	Link to Source
2.0.0	~5.5.0 | ~5.6.0 | ~7.0.0	composer.json
2.0.18	~5.5.0 | ~5.6.0 | ~7.0.0	composer.json
2.1.0	~5.6.0 | 7.0.2 | ~7.0.6	composer.json
2.1.18	~5.6.5 | 7.0.2 | 7.0.4 | ~7.0.6 | ~7.1.0	composer.json
2.2.0	~7.0.2 | 7.0.4 | ~7.0.6 | ~7.1.0	composer.json
2.2.11	~7.0.13 | ~7.1.0 | ~7.2.0	composer.json
2.3.0	~7.1.3 | ~7.2.0	composer.json
2.3.6-p1	~7.1.3 | ~7.2.0 | ~7.3.0	composer.json
2.4.0	~7.3.0 | ~7.4.0	composer.json
2.4.2	~7.3.0 | ~7.4.0	composer.json

Workflow for Magento 2 Settings

I found this treasure hidden in the Magento Github repo describing succinctly on how to work with Magento's configuration files, namely `app/etc/config.php` and `app/etc/env.php`. Reposting it here so that I can refer to it back more easily in the future.

In summary `app/etc/config.php`is shared across all instances but `app/etc/env.php`only exists individually on each instance. The first is part of your code repository the latter isn't.

Here is Juan Alonso's example of a workflow on a real project:

1. When creating the project we dump all settings with `app:config:dump scopes themes` That creates the needed settings for scopes and themes but skips system core config data

2. We also add shared settings needed for PRD and Build environments

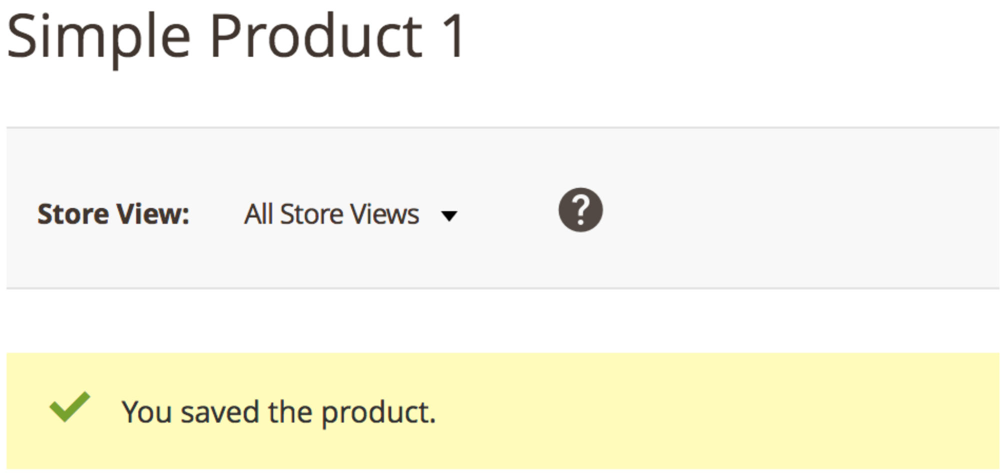
```
bin/magento config:set --lock-config dev/js/merge_files 1

bin/magento config:set --lock-config dev/css/merge_css_files 1

bin/magento config:set --lock-config dev/static/sign 1
```

3. As project evolves, we add a new stores and themes, so we need to update our config.php settings. `bin/magento app:config:dump scopes themes`

4. At the same time, if we ever need to share system settings on all environments, we can use the command `bin/magento config:set --lock-config` without needing to dump all hundreds of settings into the config.php

Displaying Complex Error Messages in Magento 2

When an end user has taken an action in any application, it's a good idea to provide the user feedback on that action.
So for example if the save product button has been clicked in Magento 2 you would expect to see something like this:


Simple Product 1

Store View: All Store Views

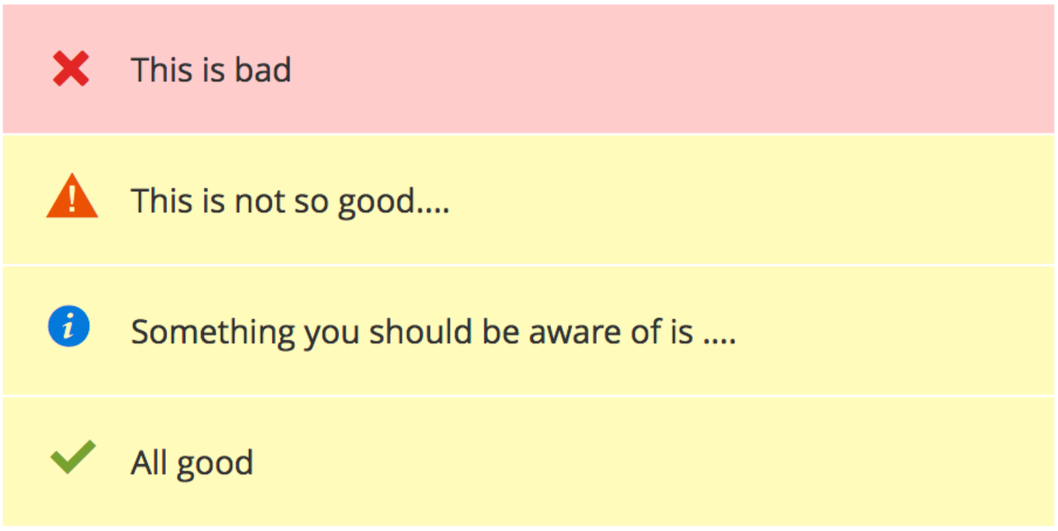
You saved the product.

If you are writing custom code for Magento, it's fairly straightforward to add your own messages to be displayed to the user. Usually you would do this in your own controller code - the context object provides you with access to the `\Magento\Framework\Message\ManagerInterface via $context->getMessageManager()` as the framework provided means to do so.

Out of the box the following messages can be used:

```
$this->messageManager->addSuccessMessage(__('All good'));

$this->messageManager->addNoticeMessage(__('Something you should be aware of is ....'));

$this->messageManager->addWarningMessage(__('This is not so good....'));

$this->messageManager->addErrorMessage(__('This is bad'));
```

which will then render to the user as:


This is bad

This is not so good....

Something you should be aware of is ....

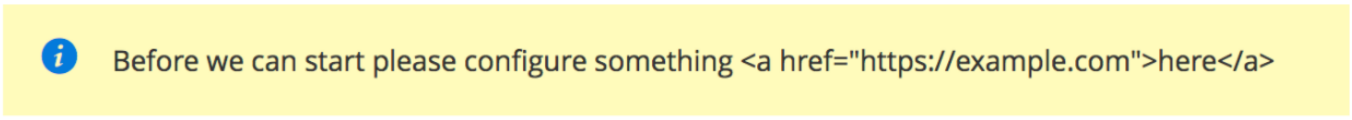
All good

This is usually all you need in user interaction.

However sometimes you need that little bit more from those messages, for example you might want to include an URL directly in the message itself.
Your first inclination probably looks similar to mine and you end up doing:

```
$this->messageManager->addNoticeMessage(__('Before we can start please
configure something <a href="https://example.com">here</a>'));
```

but we soon find out that all output gets automatically escaped:


Before we can start please configure something <a href="https://example.com">here</a>

Luckily for us there is an inbuilt solution for us to use custom messages. In the `\Magento\Framework\Message\ManagerInterface` there is an extra set of methods which all include Complex in their name:

```
/* @see \Magento\Framework\Message\ManagerInterface */

public function addComplexSuccessMessage($identifier, array $data = [], $group = null);

public function addComplexErrorMessage($identifier, array $data = [], $group = null);

public function addComplexWarningMessage($identifier, array $data = [], $group = null);

public function addComplexNoticeMessage($identifier, array $data = [], $group = null);
```

To make use of them we first need to set up our identifier via some di.xml instructions:

```
<!-- etc/adminhtml/di.xml-->
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <type name="Magento\Framework\View\Element\Message\MessageConfigurationsPool">
        <arguments>
            <argument name="configurationsMap" xsi:type="array">
                <item name="foomanExample" xsi:type="array">
                    <item name="renderer" xsi:type="const"
                       >\Magento\Framework\View\Element\Message\Renderer\BlockRenderer::CODE</item>
                    <item name="data" xsi:type="array">
                        <item name="template" xsi:type="string"
                           >Fooman_Example::messages/foomanExample.phtml</item>
                    </item>
                </item>
             </argument>
         </arguments>
     </type>
</config>
```

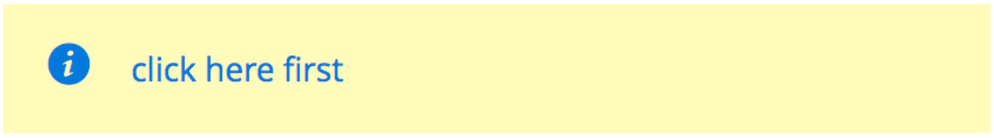
The template file could then look like this

```
/* view/adminhtml/templates/messages/foomanExample.phtml */

<a href="<?php echo $block->escapeUrl($block->getData('url'))?>"><?php echo $block->escapeHtml(__('click here first')) ?></a>
```

and last but not least we need to invoke this message and pass in the URL information as part of the data array:

```
$this->messageManager->addComplexNoticeMessage(

    'foomanExample',

    [

        'url' => $this->_helper->getUrl('to/our/route')

    ]

    );
```

And we have our intended end result:


click here first

As a quick reminder do not output user input without properly sanitizing the input and escaping the output. The various `$block->escapeX` methods like `$block->escapeHtml()` and `$block->escapeUrl()` are your friends.

Working Faster with bin/magento

The `bin/magento` command is the entry point to working with Magento 2's command line tasks. Prompted by a couple of recent tweets I thought I quickly write down a couple of tips to speed up working with it.

First up, depending on how your system is configured, you are either able to use `bin/magento` directly or you need to type `php -f bin/magento` (or even something like `php70 -f bin/magento` if you have multiple versions of php running at the same time). This gets rather cumbersome to type out all the time...

Enter command aliases.

You can provide an alias to any command simply by running:

```
alias m2='bin/magento'
```

(or whatever else command you need to run - see above).

Now instead of typing out `bin/magento` all the time we can just type `m2` followed by whatever task we want to execute, so for example `m2 cache:clear`. Read more on aliases for example here to make this permanent by adding this to your ~/.bashrc file.

We can take this a step further as well. The `bin/magento` command only works when executed from the root directory of your site. We can switch to this folder as part of the aliased command:

```
alias m2='cd /var/www && bin/magento'
```

Now we can type `m2` from anywhere and it will take us to the Magento 2 folder as the first step and then runs the Magento command.

There is one further improvement we can make. It is important to not mix up different levels of permissions as that can break things. To avoid the `bin/magento` command being run with a different user to the one owning the file we add an additional safe guard via an if clause:

```
alias m2='cd /var/www && if [ `whoami` == `stat -c '%U' bin/magento` ]; then bin/magento "$@"; else echo "bin/magento should be run as user `stat -c '%U' bin/magento`";fi'
```

Next the task arguments after `bin/magento` can all be shortened as well. So instead of typing out `bin/magento cache:clean` you can just use `bin/magento c:c`. You can abbreviate any task like this. However you might come across an error message like:

```
$:/var/www$ bin/magento i:r

[Symfony\Component\Console\Exception\CommandNotFoundException]

Command "i:r" is ambiguous (indexer:reindex, indexer:reset).
```

in the above case we abbreviated too much and the system can't determine if we wanted to use indexer:reindex or indexer:reset. So we need to make sure that it can differentiate between the two by either using `$:/var/www$ bin/magento i:rei` or `$:/var/www$ bin/magento i:res`

The alias can be combined with the shortened arguments as well which adds up to quite a few less characters to type:

```
php -f bin/magento cache:clean

m2 c:c
```

Instead of using abbreviations you can also use command completion instead of typing out every task you can hit [TAB]. Please see the project here.

Working with Warden

There are lots of different options around to set up your local development environment and I'd hazard a guess no two development environments are the same. Most of my Magento development work I do is on a Mac so I have come across Homebrew based set ups, in combination with or without Valet+ and Docker based approaches. For Docker the most mature and widely used one is the one by Mark Shust.

Unfortunately for me when I started M2 development none of these were around so I made do with my own inhouse solution based on Docker. It is fair to say that my set up took a while to get right and I am still always on the look out to see if there is something I can improve. In particular one of my requirements is that I am able to run multiple Magento versions in parallel and I need to be able to start with a freshly installed instance often. This has ruled out a lot of the approaches that include syncing as the initial sync time takes a while and options that work better for single instances.

With that preface out of the way I was excited to come across a new command line tool by David Alger called Warden to aid with Docker based M2 development. The part which immediately appealed to me was that it has two parts to it.

General Docker Tools

Warden comes with a set of tools that make working with containers easier:

- Provides own certificate authority to provide TLS
- DNS routing via Traefik
- Portainer (nice GUI to see more details into what Docker is doing)
- plus a few more items that I haven't looked into further yet like ssh forwarding.

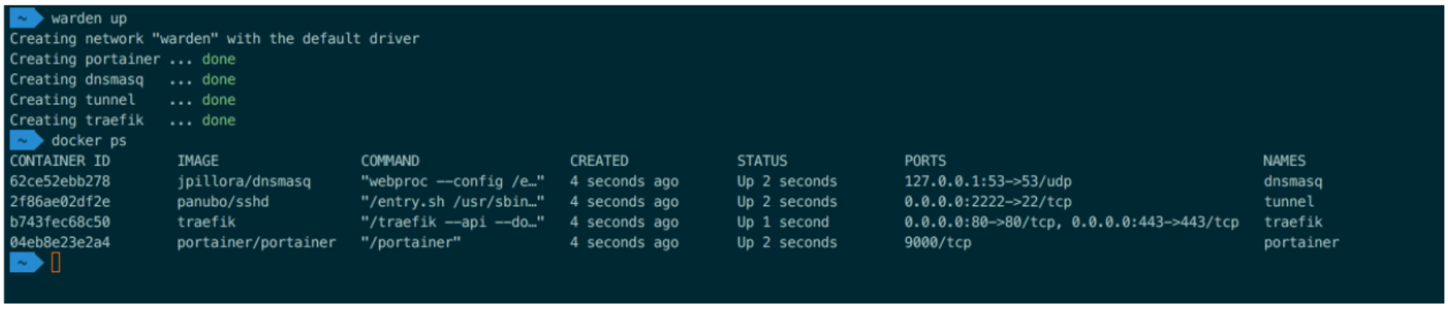
Project Based Environments

With the first part taking care of general "plumbing" tasks Warden then provides the ability to spin up specific project based environments. It comes with a pre-installed Magento 2 template that even smoothes out some of the differences between a Mac vs Linux workflow.

What made it especially appealing for me is that integrating our existing images with 1.) only required adding a few docker labels and allowed me to retire some of the manual set up I used for faking DNS entries (dnsmasq) and routing them back to a local container (nginx-proxy with docker-gen). I then was able to use 2.) in parallel with our own custom extension focused images.

How Does It Work?

After following the installation instructions the first command to run is `warden up` which spins up the mentioned infrastructure. You can see the newly running containers with `docker ps:`

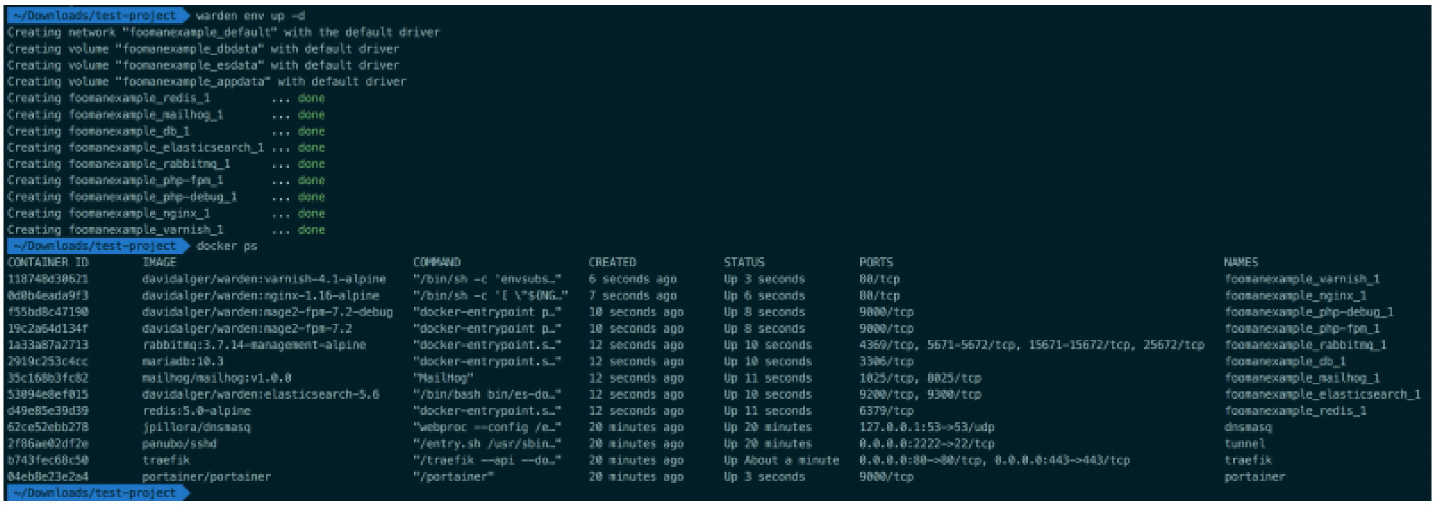


```
~ warden up
Creating network "warden" with the default driver
Creating portainer ... done
Creating dnsmasq   ... done
Creating tunnel    ... done
Creating traefik   ... done
~ docker ps
CONTAINER ID   IMAGE                 COMMAND                  CREATED         STATUS         PORTS                                       NAMES
62ce52ebb278   jpillora/dnsmasq      "webproc --config /e…"   4 seconds ago   Up 2 seconds   127.0.0.1:53->53/udp                        dnsmasq
2f86ae02df2e   panubo/sshd           "/entry.sh /usr/sbin…"   4 seconds ago   Up 2 seconds   0.0.0.0:2222->22/tcp                        tunnel
b743fec68c50   traefik               "/traefik --api --do…"   4 seconds ago   Up 1 second    0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp    traefik
04eb8e23e2a4   portainer/portainer   "/portainer"             4 seconds ago   Up 2 seconds   9000/tcp                                    portainer
~
```

The following links provide useful information while Warden is running:

- Traefik https://traefik.warden.test/ - shows you which containers are responding to what domain/port
- Portainer https://portainer.warden.test - overview of images and running containers
- dnsmasq https://dnsmasq.warden.test/ - if you need to debug DNS or want to change the dnsmasq config

From here we can use the included Magento 2 template to provide us with an environment that comes with php, database, varnish, rabbitmq, redis and mailhog. All we need to get all this up and running is to run `warden env-init foomanexample magento2` from your project root folder, followed by `warden sign-certificate foomanexample.test` to trust the new domain `foomanexample.test` and finally `warden env up -d`. What happens in the background here is that Warden checks your local configuration in the `.env` file and the `.warden` folder and then combines it with the included docker-compose templates. You can see the outcome of the behind the scenes work by running `docker ps` once more to show all the different services now available via a docker container.



```
~/Downloads/test-project  warden env up -d
Creating network "foomanexample_default" with the default driver
Creating volume "foomanexample_dbdata" with default driver
Creating volume "foomanexample_esdata" with default driver
Creating volume "foomanexample_appdata" with default driver
Creating foomanexample_redis_1         ... done
Creating foomanexample_mailhog_1       ... done
Creating foomanexample_db_1            ... done
Creating foomanexample_elasticsearch_1 ... done
Creating foomanexample_rabbitmq_1      ... done
Creating foomanexample_php-fpm_1       ... done
Creating foomanexample_php-debug_1     ... done
Creating foomanexample_nginx_1         ... done
Creating foomanexample_varnish_1       ... done
~/Downloads/test-project  docker ps
CONTAINER ID   IMAGE                                      COMMAND                  CREATED          STATUS              PORTS                                                 NAMES
118748d30621   davidalger/warden:varnish-4.1-alpine       "/bin/sh -c 'envsubs…"   6 seconds ago    Up 3 seconds        80/tcp                                                foomanexample_varnish_1
0d0b4eada9f3   davidalger/warden:nginx-1.16-alpine        "/bin/sh -c '[ \"$ONG…"  7 seconds ago    Up 6 seconds        80/tcp                                                foomanexample_nginx_1
f55bd8c47190   davidalger/warden:mage2-fpm-7.2-debug      "docker-entrypoint p…"   10 seconds ago   Up 8 seconds        9000/tcp                                              foomanexample_php-debug_1
19c2a64d134f   davidalger/warden:mage2-fpm-7.2            "docker-entrypoint p…"   10 seconds ago   Up 8 seconds        9000/tcp                                              foomanexample_php-fpm_1
1a33a87a2713   rabbitmq:3.7.14-management-alpine          "docker-entrypoint.s…"   12 seconds ago   Up 10 seconds       4369/tcp, 5671-5672/tcp, 15671-15672/tcp, 25672/tcp   foomanexample_rabbitmq_1
2919c253c4cc   mariadb:10.3                               "docker-entrypoint.s…"   12 seconds ago   Up 10 seconds       3306/tcp                                              foomanexample_db_1
35c168b3fc82   mailhog/mailhog:v1.0.0                     "MailHog"                12 seconds ago   Up 11 seconds       1025/tcp, 8025/tcp                                    foomanexample_mailhog_1
53894e8ef015   davidalger/warden:elasticsearch-5.6        "/bin/bash bin/es-do…"   12 seconds ago   Up 10 seconds       9200/tcp, 9300/tcp                                    foomanexample_elasticsearch_1
d49e85e39d39   redis:5.0-alpine                           "docker-entrypoint.s…"   12 seconds ago   Up 11 seconds       6379/tcp                                              foomanexample_redis_1
62ce52ebb278   jpillora/dnsmasq                           "webproc --config /e…"   20 minutes ago   Up 20 minutes       127.0.0.1:53->53/udp                                  dnsmasq
2f86ae02df2e   panubo/sshd                                "/entry.sh /usr/sbin…"   20 minutes ago   Up 20 minutes       0.0.0.0:2222->22/tcp                                  tunnel
b743fec68c50   traefik                                    "/traefik --api --do…"   20 minutes ago   Up About a minute   0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp              traefik
04eb8e23e2a4   portainer/portainer                        "/portainer"             20 minutes ago   Up 3 seconds        9000/tcp                                              portainer
~/Downloads/test-project
```

If you access https://app.foomanexample.test Warden will provide the DNS entry and then route this via the Traefik container to the Varnish container, which in turn will talk to the nginx back-end to provide the requested resource.

If you are on a Mac Warden includes Mutagen.io to sync code changes in and out of the container. Start it with `warden sync start`. I have had to re-start the syncing a few times (simply re-issue warden sync start) which I am not sure where due to spinning up and down the images fairly quickly or if waking the laptop from sleep had something to do with it. During normal operation the sync is super quick and does not get in the way of developing.

For a full example walk through of installing Magento 2 configured to use all the services included please see the documentation here.

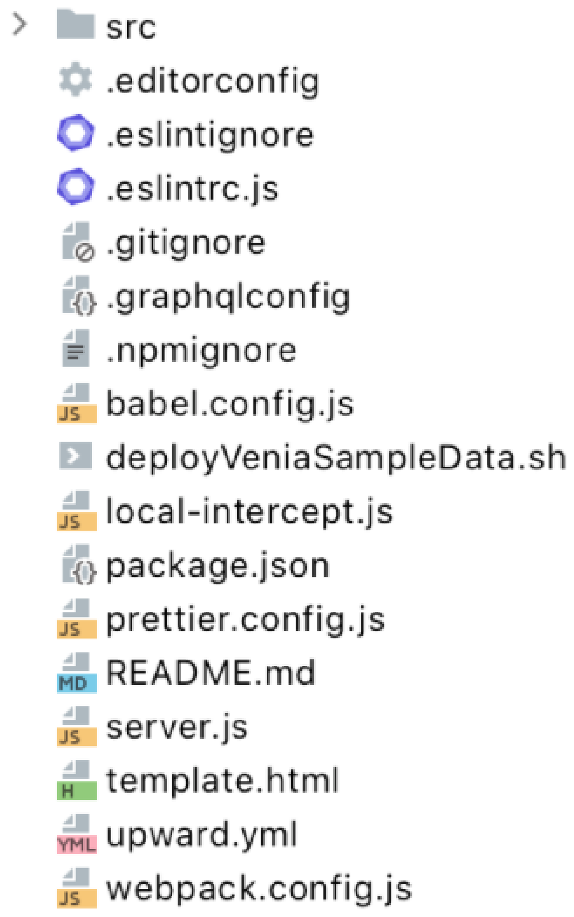
Upgrading a Scaffolded PWA Studio Project

When working with PWA Studio you have two ways to set up your project:

1. Clone the repository
2. Use the scaffold command`yarn create @magento/pwa` to create your starting project

When using 1. getting the latest code base is usually just a git pull / merge away. However with 2. this is not as straightforward. Essentially when using a create command, as is common in the React world, you create a snapshot of the project at the time you run the command and future updates are your responsibility to integrate.

If for example you started in August 2020 you would have created a project based on PWA Studio 7.



```
> src
  .editorconfig
  .eslintignore
  .eslintrc.js
  .gitignore
  .graphqlconfig
  .npmignore
  babel.config.js
  deployVeniaSampleData.sh
  local-intercept.js
  package.json
  prettier.config.js
  README.md
  server.js
  template.html
  upward.yml
  webpack.config.js
```

Before running any `yarn install` the folder content would resemble the package content of venia-concept here. However this is not 100% accurate as some additional processing is run from the source to what gets published via npm - see the buildpack script here.

create-pwa Downsides

The major downside to using the `create-pwa` approach is that once a new version of PWA Studio gets released how to know what to update? The main areas are:

Package Updates
How do we know for example that PWA Studio 7 ships `@magento/venia-ui` at version 4.0.0 and version 5.0.0 for PWA Studio 8? At the same time the apollo client was updated from version 2.x to 3.x And this is not a one off either as, for example, PWA Studio 9 includes the jump to React 17.

Webpack config changes
The main `webpack.config.js` is also a file that is in your project. Over the time I have been tracking this project there have been various changes and improvements made to this file.

Service Worker improvements
The service worker which is set up via the code under `src/ServiceWorker` is also not set in stone and has seen a number of changes.

create-pwa always uses the latest versions
To my knowledge it is not possible to instruct yarn to use a different version for the create command to essentially go back in time to easily create a diff between the resulting files.

Lots of diffing
In summary the above issues lead to a few headaches when trying to track the upstream PWA Studio project to benefit from the improvements made since you started. If you tried to reconstruct the history of individual files from the source repository I think this becomes unmanageable.

The solution

To be able to create a proper diff between versions for projects started with `create-pwa` is to re-run the same packaging command the PWA Studio team uses to create the final package. This is exactly what I have done in our PWA Studio Starter repository for the last 3 PWA Studio releases.

You can check the changes in the below links here:

- Changes from PWA Studio 7.0.0 to 8.0.0
- Changes from PWA Studio 8.0.0 to 9.0.1

or if you are starting a new project you can also clone the repository directly (we also create a weekly package if you would like to track the upstream progress more closely).

Magento 2 Handling of Order Timezone

Triggered by a recent discussion on Slack and noticing this latest Pull Request to fix some timezone handling specific code in the Magento 2 framework (Github PR) it seems to confirm a long held suspicion that something in the Magento framework timezone handling was not quite working the way I believe it should (even going back to M1 days). Below is an approach similar to what I have been using to translate for example the order date to the timezone of the store. It would be able to handle any Magento model that descends from `\Magento\Framework\DataObject` and has `getCreatedAt()` and `getStoreId()` methods.

```
class CreationTimeAtTimezone


  /**
  * @var \Magento\Framework\App\Config\ScopeConfigInterface
  */
  private $scopeConfig;

  public function __construct(
     \Magento\Framework\App\Config\ScopeConfigInterface $scopeConfig
  ) {
     $this->scopeConfig = $scopeConfig;
  }

  public function getCreatedAtStore(\Magento\Framework\DataObject $object, $format = 'Y-m-d')
  {
     $datetime = \DateTime::createFromFormat('Y-m-d H:i:s', $object->getCreatedAt());
     $timezone = $this->scopeConfig->getValue(
         'general/locale/timezone',
         \Magento\Store\Model\ScopeInterface::SCOPE_STORE,
         $object->getStoreId()

     );
     if ($timezone) {
         $storeTime = new \DateTimeZone($timezone);
         $datetime->setTimezone($storeTime);

     }
     return $datetime->format($format);

  }

}
```

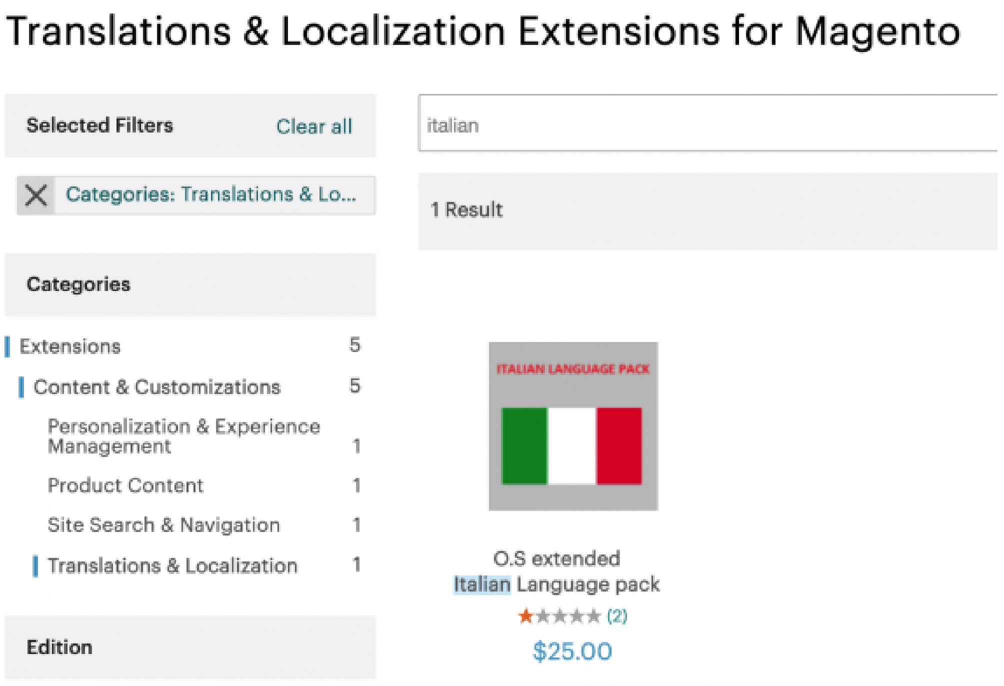
Installing Community Language Packs

Magento has always been used around the world with the community providing translations for different locales.

With Magento 2 this is no different and the translation efforts can be found on Crowdin. What is less obvious is how do you actually use those translations? When installing Magento 2 the following language packs are installed out of the box:

- language-de_de
- language-en_us
- language-es_es
- language-fr_fr
- language-nl_nl
- language-pt_br
- language-zh_hans_cn

For any other languages we are unfortuntately sent on a wild goose chase. Let's say for example we would like to add Italian. The devdocs for adding a language point us to a category on Marketplace. To my surprise searching for Italian only brings up a commercial offering with less than flattering reviews.


Translations & Localization Extensions for Magento
Selected Filters
Clear all
Categories: Translations & Lo...
italian
1 Result
Categories
Extensions 5
Content & Customizations 5
Personalization & Experience Management 1
Product Content 1
Site Search & Navigation 1
Translations & Localization 1
ITALIAN LANGUAGE PACK
O.S extended Italian Language pack
(2)
$25.00
Edition

Found Them

Fortunately for us Magento's community engineering team is turning the Crowdin translations into installable translations packs. Those efforts can be seen under this Github account. And what is even better those packages get in turn published via Packagist.

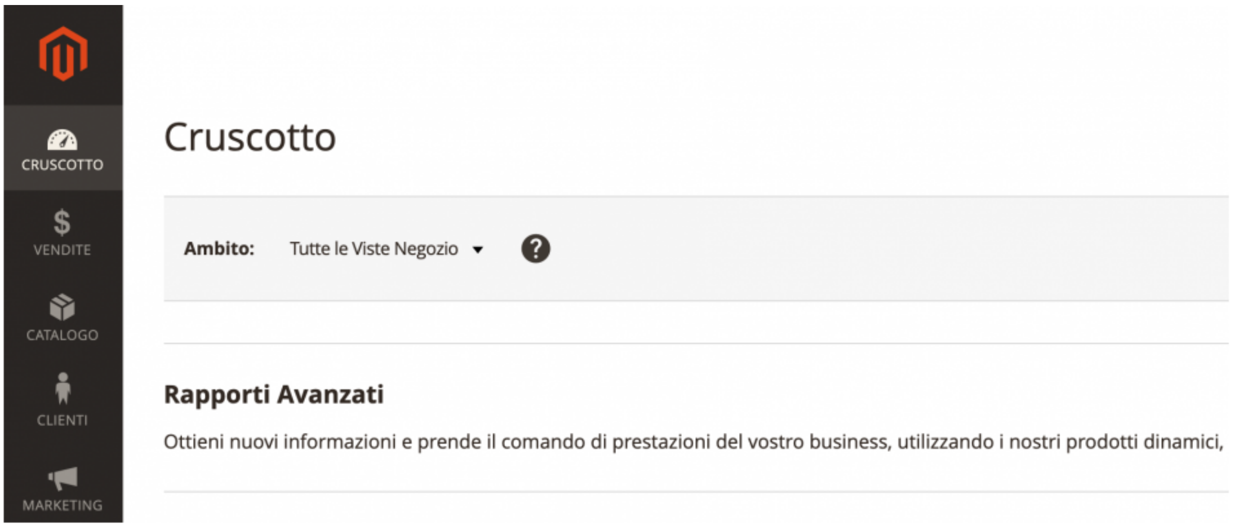
The complete list can be viewed here.

The package name follows the format `"community-engineering/language-" + strtolower(name of locale)`

So for our Italian example we would have:

```
composer require community-engineering/language-it_it

bin/magento cache:clean
```

And after changing the locale of the admin ui in the admin user's setting we are welcomed with a dashboard in Italian:



Cruscotto

Ambito: Tutte le Viste Negozio

Rapporti Avanzati

Ottieni nuovi informazioni e prende il comando di prestazioni del vostro business, utilizzando i nostri prodotti dinamici,

Switching to the Unified Magento Coding Standard

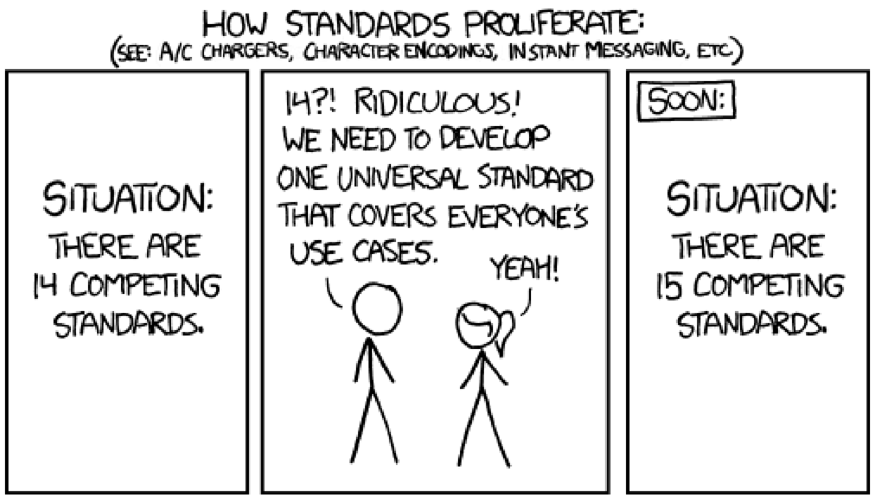
When working with Magento 2 depending on where you look there are different coding standards one might pick to follow:

- PSR-2 (Magento is part of PHP-FIG after all)
- MEQP-2 (Marketplace)
- ECG-2 (Expert consulting group)
- Standard contained in the Magento 2 code base

This is less than ideal and often leads to a situation of "do as we say, not do as we do" (not a big fan) or using a lowest denominator. Even ExtDN decided to jump into the mix to work out some workable compromise on the above.

I am pleased to see that Magento took the initiative and has been working on a unified standard that can be used across the entire Magento ecosystem. These efforts have been led by Lena Orobei and have resulted in the first releases of the Magento 2 Coding Standard.

There is of course always the danger of standard proliferation when introducing any new standard but I am confident this has a good chance of retiring all mentioned ones apart from PSR-2. What makes me hopeful is that any adopted rules are already run against the Magento 2 code base and issues are being identified and addressed as part of the rollout.


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Getting Started
So how do we get started? First we need to install the standard, which we can add as a development requirement to our project

```
composer require magento/magento-coding-standard:* --dev
```

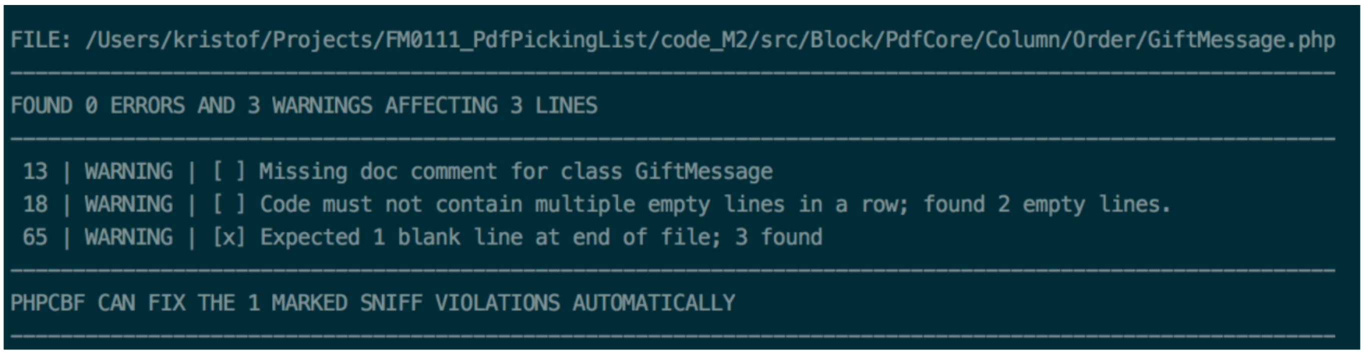
From there we need to let PhpCodeSniffer (got automatically installed with the above) know about the new standard:

```
vendor/bin/phpcs --config-set installed_paths ../../magento/magento-coding-standard/
```

And from there we can use it against any code directory we want to statically test

```
vendor/bin/phpcs --standard=Magento2 src
```

and we will likely see a few things mentioned the first time we run it.



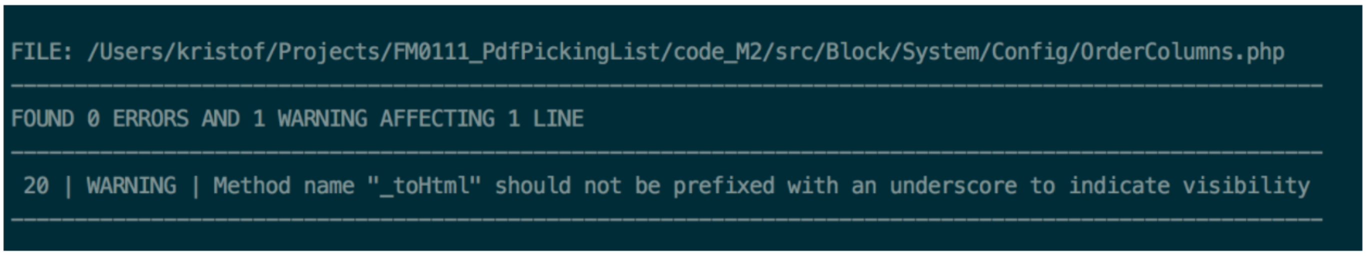
```
FILE: /Users/kristof/Projects/FM0111_PdfPickingList/code_M2/src/Block/PdfCore/Column/Order/GiftMessage.php
-----------------------------------------------------------------------------------------------------------
FOUND 0 ERRORS AND 3 WARNINGS AFFECTING 3 LINES
-----------------------------------------------------------------------------------------------------------
 13 | WARNING | [ ] Missing doc comment for class GiftMessage
 18 | WARNING | [ ] Code must not contain multiple empty lines in a row; found 2 empty lines.
 65 | WARNING | [x] Expected 1 blank line at end of file; 3 found
-----------------------------------------------------------------------------------------------------------
PHPCBF CAN FIX THE 1 MARKED SNIFF VIOLATIONS AUTOMATICALLY
-----------------------------------------------------------------------------------------------------------
```

From there we can hopefully work through these to remove them. Also take a look at running `vendor/bin/phpcbf --standard=Magento2 src` as that can automate some of the fixes.

There is More Than 1 Standard
I like to keep my code following PSR-2 as well. As that standard is one of the de-facto standards in the php world it comes pre-installed with PhpCodeSniffer and we can run it with

```
vendor/bin/phpcs --standard=PSR2 src
```

Working with Magento this will likely lead to some rule violations which we can't do much about (short of changing M2 whole sale, hello M1 legacy).



```
FILE: /Users/kristof/Projects/FM0111_PdfPickingList/code_M2/src/Block/System/Config/OrderColumns.php
---------------------------------------------------------------------------------------------------------
FOUND 0 ERRORS AND 1 WARNING AFFECTING 1 LINE
---------------------------------------------------------------------------------------------------------
 20 | WARNING | Method name "_toHtml" should not be prefixed with an underscore to indicate visibility
---------------------------------------------------------------------------------------------------------
```

In these cases code sniffer allows us to add exceptions via annotations or code comments.

```
// phpcs:ignore PSR2.Methods.MethodDeclaration -- Magento 2 core use

public function _toHtml()

{

    if (!$this->getOptions()) {

        $this->setOptions($this->getColumns());

    }

    return parent::_toHtml();


}
```

We should aim to adhere to the standard first before trying to exclude our code. Not all code is special and one should be prepared to provide a reason to add such an exception if this comes up in a code review. Also make the exclusion as specific and as narrow as possible, in other words you do not need to skip the whole file. If you need to find out the exact rule that triggered the warning you can run `phpcs` with the `-s flag: vendor/bin/phpcs --standard =PSR2 src -s`.

Automate It
One of the best ways to adopt a coding standard is to enforce it automatically when you check in any code. When working with git, git hooks are perfect for this. I have recently come across CaptainHook which makes configuring and sharing these hooks in your repository easier. Other tools like Grumphp can do the same for you. The aspect that appealed to me to start using CaptainHook is that I can keep the hook set up manual while I test out how this works for me before integrating this with Composer for all users as well.

```
composer require --dev captainhook/captainhook

vendor/bin/captainhook configure
```

Which will take you through a quick questionnaire. Alternatively you can supply a `captainhook.json` file in your root folder.

```
{
      "commit-msg": {
          "enabled": false,
          "actions": []
      },
      "pre-push": {
          "enabled": false,
          "actions": []
      },
      "pre-commit": {
          "enabled": true,
          "actions": [
              {
                  "action": "\\CaptainHook\\App\\Hook\\PHP\\Action\\Linting",
                  "options": [],
                  "conditions": []
              },
              {
                  "action": "vendor/bin/phpcs --config-set installed_paths ../../magento/magento-coding-standard/ && vendor/bin/phpcs --standard=Magento2 src",
                  "options": [],
                  "conditions": []

              },
          ]
      },
      "prepare-commit-msg": {
          "enabled": false,
          "actions": []
      },
      "post-commit": {
          "enabled": false,
          "actions": []
      },
      "post-merge": {

          "enabled": false,
          "actions": []
      },
      "post-checkout": {
          "enabled": false,
          "actions": []
      }
}
```

Which you can then activate with `vendor/bin/captainhook install`. From here on out any attempts to check in code with `git commit` that fails the php linting or Magento coding standard will get rejected, requiring you to apply a fix for it.

Changing to use the composer plugin instead will integrate with `composer install` or `update` and automate the hook activation`composer require --dev captainhook/plugin-composer` .

The standard is being collaboratively developed on github and you can check out its progress and underlying discussions here.

FOOMAN

Great Magento Developers
Need Great Tools.

For more Magento 2 developer
tips, visit the Fooman blog.

FOOMAN BLOG >