Drupal Development

Philip Norton hashbangcode.com @hashbangcode @philipnorton42

Drupal Development

- Extending Drupal's functionalty to do whatever you want it to.
- Covers Modules, Themes, Install Profiles.

Setting Up Drupal For Development

Setting Up Drupal

- Lots of options exist to ease development in Drupal.
- This includes turning off the Drupal cache, forcing autodiscovery on every page load and preventing permission hardening.
- Adding these options makes Drupal development easier.

Setting Up Drupal

- Drupal has a `site/example.settings.local.php` file.
- Copy this to `site/default/settings.local.php`.

```
cp site/example.settings.local.php site/default/settings.local.php
```

Setting Up Drupal

- Uncomment the following from the bottom of the `sites/default/settings.php` file.

```
if (file_exists($app_root . '/' . $site_path . '/settings.local.php'))
  include $app_root . '/' . $site_path . '/settings.local.php';
}
```

Setting Up Drupal

- The settings.local.php file will also include a `sites/development.services.yml` file.
- This turns on cacheability headers and turns allows backend cache classes to be pucked up.
- The file looks like this.

```yaml
parameters:
  http.response.debug_cacheability_headers: true
services:
  cache.backend.null:
    class: Drupal\Core\Cache\NullBackendFactory
```

Setting Up Drupal

- Add Twig debugging and auto reload are configured in the .

```
parameters:
  twig.config:
    debug: true
    auto_reload: true
    cache: false
  http.response.debug_cacheability_headers: true
services:
  cache.backend.null:
    class: Drupal\Core\Cache\NullBackendFactory
```

Setting Up Drupal

- Twig debugging comments.

```
<!-- THEME DEBUG -->
<!-- THEME HOOK: 'block' -->
<!-- FILE NAME SUGGESTIONS:
   * block--umami-account-menu.html.twig
   * block--system-menu-block--account.html.twig
   x block--system-menu-block.html.twig
   * block--system.html.twig
   * block.html.twig
-->
<!-- BEGIN OUTPUT from 'core/profiles/demo_umami/themes/umami/templates
<nav role="navigation" aria-labelledby="block-umami-account-menu-menu"
```

Setting Up Drupal

- Ensuring the setting has taken.

```
drush php:eval "var_export(\Drupal::getContainer()
   ->getParameter('twig.config'));"
```

Setting Up Drupal

- Drupal will check the permissions of your settings.php files and ensure they are secure.
- To turn this off make sure this setting is enabled.

```
$settings['skip_permissions_hardening'] = TRUE;
```

Setting Up Drupal

- To turn off the permanent caches uncomment any line that looks like this.

```
$settings['cache']['bins']['x'] = 'cache.backend.null';
```

Try it!

- Turn on Drupal debug settings.

Devel

Devel

- The Devel module is a good way of finding out more about the current state of Drupal.
- The Web Profiler is a sub module that can be used to drill into routes, database queries, hooks, cache systems and other things.

Try it!

- Install Devel and Web Profiler.
- See it in action.

Drupal Module Development

What Is A Module?

- Adds a feature to a site.
- Can be turned on or off.
- Can define extra functionality or hook into and override other parts of Drupal.

Types Of Module

- **Core** - Included in Drupal itself.
- **Contributed** - Any third party module you install. Referred to as "contrib".
- **Custom** - Any module you build yourself.

Writing A Module

The *.info.yml File

- Contains information about the module including what it does and what version on Drupal it is compatable with.
- In YAML format.
- The bare minimum required for a Drupal module to be picked up.

mymodule.info.yml

```
name: 'My Module'
type: module
description: 'My amazing module.'
core_version_requirement: ^8 || ^9
```

Documentation

Documentation

- A module should include a readme file.
- This should include:
    - Module functionality and how to use it.
    - Configuration options.
    - Available hooks/events.

Documentation

- Add a readme file to the top level of your module.
- Markdown format is preferred.
- Can also use plain text.

```
README.md
```

README template: https://bit.ly/3KaXi5Q

README Template

- Preferred on larger files.

```
CONTENTS OF THIS FILE
---------------------

 * Introduction
 * Requirements
 * Recommended modules
 * Installation
 * Configuration
 * Troubleshooting
 * FAQ
 * Maintainers
```

Hooks

The simplest building block of any module.

What Is A Hook?

Hooks allow you to:

- Alter forms.
- Alter theme elements before rendering.
- React to events.
- Register plugins and templates.

Any module can define custom hooks.

Some Popular Hooks

- `hook_form_alter($form, $fotm_state, $id)`
- `hook_theme($existing, $type, $theme, $path)`
- `hook_preprocess_page(&$variables)`
- `hook_theme_suggestions_alter(&$suggestions, $variables, $hook)`
- `hook_node_insert($entity)`
- `hook_node_update($entity)`
- `hook_update_9001(&$sandbox)`

Naming Hooks

- Hooks are named after the module they appear in.

```
hook_form_alter()
```

Becomes:

```
mymodule_form_alter()
```

- The hook_form_alter() hook is called every time a form is created.

Naming Hooks

- Some hooks also change their name based on context.

```
hook_node_insert($entity)
```

Can also be:

```
hook_user_insert($entity)
```

When detecting users being created.

Example Hook

Use hook form alter to alter a form.

```php
use Drupal\Core\Form\FormStateInterface;

function mymodule_hook_form_alter(
  &$form,
  FormStateInterface $form_state,
  $form_id) {
  if ($form_id == 'node_article_form') {
    $form['title']['widget'][0]['value']['#default_value'] = t('title')
  }
}
```

Try It!

- Create the file `mymodule.module`.
- Add a hook to alter a form.
- Flush caches!

Translation

Translation

- Why talk about multilingual code so early?
- It's baked into everything Drupal does. Drupal is multilingual from the start.
- You will see either t() or $this->t() a lot.
- These functions will register the translation with the Drupal translation system.

t() Usage

- To use both t() and $this->t() just pass in a string.

```
$translated = t('String');
```

```
$translated = $this->t('String');
```

- Best practice is to pass it directly into where it is needed, rather than store in a variable.

Passing Arugments

Pass escaped output (should be your default choice).

```
$t = t('Value = @value', ['@value' => '123']);
```

Wrap in <em> tags.

```
$t = t('Value = %value', ['%value' => '123']);
```

Escape (used for URLs)

```
$t = t('<a href=":url">@variable</a>',
  [':url' => $url, '@variable' => $variable]);
```

Controllers

Controllers

- Add an action for a particular **Route**.
- Parameters can be passed to the controller.
- Should return an array of content ready to be rendered or a response object.
- Multiple routes can use the same controller.

Routes

- All controllers need a route.
- This tells Drupal what controller to use when a path is requested.
- Defined in a *.routes.yml file.

Routes

Create a file at mymodule.routing.yml.

```
mymodule.controller_action:
  path: '/mycontroller/action'
  defaults:
    _controller: '\Drupal\mymodule\Controller\MyController::action'
    _title: 'My Controller'
  requirements:
    _access: 'TRUE'
```

Controller

A basic controller looks like this.

```php
<?php

namespace Drupal\mymodule\Controller;

use Drupal\Core\Controller\ControllerBase;

class MyController extends ControllerBase {
  public function action() {
    // return a render array or a new response object.
  }
}
```

Controller Return A Response

A basic controller looks like this.

```php
namespace Drupal\mymodule\Controller;

use Drupal\Core\Controller\ControllerBase;
use Symfony\Component\HttpFoundation\Response;

class MyController extends ControllerBase {
  public function action() {
    return new Response('Response.');
  }
}
```

Different Types Of Response Objects Exist

- **Response** - Text based response.
- **HtmlResponse** - A HTML response.
- **JsonResponse** - JSON response.
- **XmlResponse** - XML response.
- **CacheableResponse** - A response that contains Drupal cache metadata.

Try It!

- Create a route.
- Add a controller for the route.
- Return a response object.

Hint: Some cache clearing may be needed.

Render Arrays

Render arrays are a hierarchical structure of elements that Drupal will convert into markup.

You can inject raw markup into render arrays, but it's generally best practice to use themes to render HTML.

Render Arrays

This render array:

```
$build = [];
$build['description'] = [
  '#type' => 'html_tag',
  '#tag' => 'p',
  '#value' => $this->t('Some description.'),
];
return $build;
```

Will become:

```
<p>Some description.</p>
```

Render Arrays

This render array:

```php
$build = [];
$build['list'] = [
  '#theme' => 'item_list',
  '#items' => ['Item 1', 'Item 2'],
];
return $build;
```

Will become:

```html
<ul><li>Items 1</li><li>Item 2</li></ul>
```

Try It!

- Change your controller to return a render array.

Hint: item_list, html_tag.

Menu Links

Menu Plugins

- You can inject menu items into Drupals menu system.
- Stored in the `*.links.menu.yml` file.
- These menu items are not editable.

```
mymodule.controller_action:
  title: 'MyModule Controller'
  description: 'A controller with an action.'
  route_name: mymodule.controller_action
  parent: system.admin
```

Menu link is created under /admin.

Try it!

- Create a route.
- Create a controller to listen to that route.
- Return some content.
- Add a menu plugin to the controller.

Passing Parameters To Routes

- This is known as adding a wildcard to a route.

```
mymodule.controller_action:
  path: '/mycontroller/action/{parameter}'
  defaults:
    _controller: '\Drupal\mymodule\Controller\MyController::action'
    _title: 'My Controller'
  requirements:
    _access: 'TRUE'
```

Controller With Parameter

A basic controller looks like this.

```php
<?php

namespace Drupal\mymodule\Controller;

use Drupal\Core\Controller\ControllerBase;

class MyController extends ControllerBase {
  public function action($parameter) {
    // return a render array
  }
}
```

Route Permissions

```
mymodule.controller_action:
  path: '/mycontroller/action/{parameter}'
  defaults:
    _controller: '\Drupal\mymodule\Controller\MyController::action'
    _title: 'My Controller'
  requirements:
    _permission: 'access content'
```

Try It!

- Add a parameter to your route.
- Add a parameter to your controller.
- Make it do something interesting in your controller.

Hint: Use dynamic functions like `str_repeat()`, `rand()`, `date()`, `range()`.

Forms

Forms

- In Drupal, all forms are generated using the Form API.
- It's like a render array, but for form fields.
- By default, all forms use POST.

Creating A Form

Change the route to point to a Form class.

```
mymodule.form:
  path: '/my-form'
  defaults:
    _form: '\Drupal\mymodule\Form\MyForm'
    _title: 'My Form'
  requirements:
    _access: 'TRUE'
```

Creating A Form

- Add a class to the directory `src/Form/MyForm.php`.
- Bluebrint of a form class (<em>on next slide</em>).

```php
namespace Drupal\mymodule\Form;

use Drupal\Core\Form\FormBase;
use Drupal\Core\Form\FormStateInterface;

class MyForm extends FormBase {
  public function getFormId() {
    return 'mymodule-myform';
  }

  public function buildForm(array $form,
   FormStateInterface $form_state
    ) {
    return $form;
  }

  public function submitForm(array &$form,
   FormStateInterface $form_state) {
    $this->messenger()->addStatus($this->t('Form submitted'));
  }
}
```

Creating A Form

- The form API is very like the render array, but centered around form elements.
- The most common form elements are:
  - textfield
  - radios
  - checkbox
  - checkboxes
  - select
  - submit

Creating A From

```
$form['name'] = [
  '#type' => 'textfield',
  '#title' => $this->t('Name'),
  '#required' => TRUE,
];
$form['submit'] = [
  '#type' => 'submit',
  '#value' => $this->t('Submit'),
];
```

Form Submission

```
public function submitForm(array &$form,
 FormStateInterface $form_state) {
  $name = $form_state->getValue('name');

  $this->messenger()->addStatus(
    $this->t('Submitted the name %name', ['%name' => $name]));
}
```

Form Validation

- Form validation happens in the validateForm() method (if implemented).
- If any errors are triggered then the submit handler is not called.
- Note that if you set the field to be "#required" then it will automatically get validated.

Form Validation

```
public function validateForm(array &$form,
 FormStateInterface $form_state) {
  $name = $form_state->getValue('name');

  if ($name == 'Bob') {
    // Name is Bob, trigger error!
    $form_state->setErrorByName('name', $this->t('Name is Bob. Cannot c
  }
}
```

Try it!

- Create a route for a form.
- Create a form.
- Submit the form.

Services And Dependency Injection

Resources

- #! code - Drupal 9: An Introduction To Services And Dependency Injection

Plugins

Plugins

- Provide functionality through a common interface.
- Most things in Drupal are actually plugins.
- Entity types, fields, blocks, image formats, routes are all plugins.
- You can also define custom plugins.

Plugins - Input Filter

- Simplest plugin is an input filter.
- This changes the text of a text area as it is rendered.
- The original text of the field is not altered.

```php
namespace Drupal\mymodule\Plugin\Filter;

use Drupal\filter\FilterProcessResult;
use Drupal\filter\Plugin\FilterBase;

/**
 * My amazing filter.
 * @Filter(
 *   id = "myamazingfilter",
 *   title = @Translation("My amazing filter"),
 *   description = @Translation("An amazing filter"),
 *   type = Drupal\filter\Plugin\FilterInterface::TYPE_TRANSFORM_REVERS
 * )
 */
class MyAmazingFilter extends FilterBase {
  public function process($text, $langcode) {
    $result = new FilterProcessResult($text);
    $result->setProcessedText(str_replace('foo', 'bar',
        $result->getProcessedText));
    return $result;
  }
}
```

Try it!

- Create an input filter plugin.
- Make it do something interesting.
- Assign it to an input filter format.
- Use it with some filtered content (node, block etc).

Entities

Entities

- Entities in Drupal represent "things".
- Nodes, users, comments, taxonomy terms are all entities.

Entities - Bundles

- Entites can have sub-types, called bundles.
- Bundles inherit all of the functionality of the entity.
- Think of them as extended classes.

Entities - Bundles

Entity	Bundles
Node	Articles, Basic Page
Media	Image, Video
Vocabulary	Category, Tags

Loading Entites

By ID:

```
$entity_id = 123;
$entity = \Drupal::entityTypeManager()
  ->getStorage('node')
  ->load($entity_id);
```

Loading Entities

By field value:

```
$value = 'some value';
$entity = \Drupal::entityTypeManager()
  ->getStorage('node')
  ->loadByProperties(['field_name' => $value]);
```

Loading Field Values

```
$field_value = $entity->get('field_name')->getValue()[0]['value'];
```

Creating Entities

Create a node.

```php
$node = Node::create([
  'title' => 'Article title',
  'type' => 'article',
]);

$node->save();

$newArticleId = $node->id();
```

Drupal Cache

Drupal Cache

- Drupal has a robust and dynamic cache system.
- Can be used as a static cache bin or as a dynamic cache.
- It's important to understand what the components are.
- Ideally, you want to cache as much as possible in the page.
- For anonymous users you typically want the entire page cached.

Cache Meta Data

- Added to render arrays to inform Drupal about how to cache the data.

Cache for an hour.

```
'#cache' => [
  'max-age' => 3600,
]
```

Cache for ever.

```
'#cache' => [
  'max-age' => \Drupal\Core\Cache\Cache::PERMANENT,
]
```

Cache Tags

- Cached data can be cached to show that it refernces something.
- This means that when upstream caches are cleared the tagged caches can also be cleared.
- For example, a page of content is saved. The cache of that page can be flushed from cached pages, views or anywhere else it is used.

Cache Tags

Create a cache tag for node 1 and node 2.

```
'#cache' => [
  'tags' => ['node:1', 'node:2'],
]
```

Create a cache tag for current user.

```
$cacheTags = User:load(\Drupal::currentUser()->id())->getCacheTags();
...
'#cache' => [
  'tags' => $cacheTags,
]
```

Cache Contexts

- This tells Drupal how to the data should be cached on the site.
- For example, the context "user.roles" will store the cache for each user role.

```
'#cache' => [
  'contexts' => ['user.roles', 'url.path_is_front'],
]
```

Cache Contexts

- Cache Contexts are hierarchical, so Drupal will cache the most granular variation to avoid unnecessary variations.
- For example, when caching a page per user its pointless to also cache a block on that page per user role.

Cache Methods

- Some plugins extend the CacheableDependencyInterface interface.
- This gives them access to the methods getCacheContexts(), getCacheTags(), and getCacheMaxAge().

```php
public function getCacheTags() {
  // With this when your node change your block will rebuild.
  if ($node = \Drupal::routeMatch()->getParameter('node')) {
    // If there is node add its cachetag.
    $tags = ['node:' . $node->id()]
    return Cache::mergeTags(parent::getCacheTags(), $tags);
  }
  // Return default tags instead.
  return parent::getCacheTags();
}

public function getCacheMaxAge() {
  return Cache::PERMANENT;
}

public function getCacheContexts() {
  return ['url'];
}
```

Cache API

- Get and set things from the Drupal cache.
- Integrates with cache tags if needed.

Get from cache.

```
\Drupal::cache()->get('cache_id');
```

Set data to cache.

```
\Drupal::cache()->set('cache_id', $data, $max_age, $cache_tags);
```

Cache API

```php
use Drupal\Core\Cache\Cache;

$uid = \Drupal::currentUser()->id();
$cache_id = 'something:' . $uid;

if ($data = \Drupal::cache()->get($cache_id)) {
  return $item;
}

$data = massive_calculation();
$cache_tags[] = 'uid:' . $uid;

\Drupal::cache()->set($cache_id, $data, Cache::PERMANENT, $cache_tags);

return $item;
```

Cache

- Some things (e.g. blocks) have special callback to return cache tags and cache context information.
- The methods getCacheTags() getCacheContexts() must return an array informing Drupal of the tags and contexts.

Templates

Tempaltes

- Tell Drupal about custom templates you want to use.
- Defined with a hook_theme() hook in modules or themes.

Templates

- Custom templates can be deinfed using hook_theme().

```
function my_module_theme() {
  return [
    'my_custom_tempalte' => [
      'variables' => [
        'description' => '',
        'some_list' => [],
      ],
    ],
  ];
}
```

Template

- The new hook can be used just like any other theme.

```
$build = [];
$build['content'] = [
  '#theme' => 'my_custom_tempalte',
  '#description' => $this->t('A description.'),
  '#some_list' => ['item1', 'item2'],
];
```

Template

- The custom theme needs a custom tempalte.
- The <em>templates</em> directory is the default location for templates in a module.
- Our hook will use <em>templates/my_custom_tempalte.html.twig</em>.

```
<p>{{ description }}</p>

{% for list_item in some_list %}
  {{ list_item }}
{% endfor %}
```

Try it!

- Create a hook_theme().
- Create a twig file.
- Render it in a normal render array.

Hint: Some cache clearing may be needed.

CSS & JavaScript

Asset Libraries

- CSS and JavaScript are loaded using asset libraries.
- Defined in a *.libraries.yml file.
- A library can contain both CSS and JavaScript files.
- Can collect together functionality.
- Dependencies can be used to ensure libraries are loaded together.

Define A Library

- A library file in a module.

```
some_library:
  version: 1.x
  css:
    layout:
      css/some-library-layout.css: {}
    theme:
      css/some-library-theme.css: {}
  js:
    js/some-library.js: {}
  dependencies:
    - core/jquery
```

CSS Style Types

- There are 5 types of CSS types which control how the order in which the CSS files are loaded.

base

layout

component

state

theme

Libraries Attachment

- hook_page_attachments() can attach any library to any page.

```
function mymodule_page_attachments(array &$attachments) {
  $attachments['#attached']['library'][] = 'mympdule/some_library';
  }
}
```

Try it!

- Create CSS code to change the background colour of the site.
- Create a library.
- Inject CSS into the site.

Libraries Attachment

- Attach the library to any render array.
- For example, in a controller:

```
public function action() {
  $build = [];

  $build['#attached']['library'][] = 'mympdule/some_library';

  return $build;
}
```

Try it!

- Inject the library into a controller.
- Make sure the library appears at the bottom of the page.

Hint: The *footer* setting will come in handy here.

Custom Blocks

Custom Blocks

- Add a class to src\Plugin\Block.
- Needs a @Block annotation.
- Extends Drupal\Core\Block\BlockBase.
- The build() method returns content as a render array.

Custom Block

```
namespace Drupal\mymodule\Plugin\Block;

use Drupal\Core\Block\BlockBase;

/**
 * Provides a custom block.
 *
 * @Block(
 *   id = "mymodule_custom_block",
 *   label = "MyModule Custom Block",
 *   admin_label = @Translation("MyModule Custom Block"),
 * )
 */
class ArticleHeaderBlock extends BlockBase {
  public function build() {}
}
```

Custom Block

- Implement ContainerFactoryPluginInterface to use services.
- You can then use the create()/__construct() mechanism to pull in the services needed.

```
namespace Drupal\mymodule\Plugin\Block;

use Drupal\Core\Block\BlockBase;
use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * Provides a 'Article Header' block.
 *
 * @Block(
 *   id = "hashbangcode_article_header",
 *   label = "Article Header",
 *   admin_label = @Translation("Article Header"),
 * )
 */
class ArticleHeaderBlock extends BlockBase implements ContainerFactoryP
}
```

Try it!

- Create a block.
- Output some content.
- Place the block on your site.

Configure Blocks

- The blockForm()/blockSubmit() allows configuration options to be saved to the block.

```php
public function blockForm($form, FormStateInterface $form_state) {
  $form['setting'] = [
    '#type' => 'textfield',
    '#default_value' => $this->configuration['setting'],
  ];
  return $form;
}

public function blockSubmit($form, FormStateInterface $form_state) {
  $this->configuration['setting'] = $form_state->getValue('setting');
}
```

Try it!

- Add a configuration form to your block.
- Pull out the configuration value into the block content.

Block Caches

- The methods getCacheTags() getCacheContexts() must return an array informing Drupal of the tags and contexts.

```
public function getCacheTags() {
  $node = \Drupal::routeMatch()->getParameter('node');
  return Cache::mergeTags(parent::getCacheTags(), ['node:' . $node->id(
}
```

```
public function getCacheContexts() {
  return Cache::mergeContexts(parent::getCacheContexts(), ['route']);
}
```

Upates

Updates

- Modules can provide update hooks to update older versions of the module.
- Mainly involved with updating database tables but can also be used to add defaults for new config items.

Updates

- The hook hook_update_N() is used to run updates.
- This must be placed into a mymodule.install file.

```
function mymodule_update_9001(&$sandbox = NULL) {

}
```

- Each update hook is run in sequence. 9001, 9002, 9003 etc.

Updates

- Updates can be run by visiting `update.php` or by running `drush updatedb`.
- Drush updates are preferred due to page timeouts and cli memory considerations.

Updates

- Update a configuration item.

```
function mymodule_update_9001(&$sandbox = NULL) {
  \Drupal::service('config.factory')
    ->getEditable('system.performance')
    ->set('css.preprocess', FALSE)
    ->set('js.preprocess', FALSE)
    ->save();
}
```

Updates

- Create a new table.

```php
function mymodule_update_9001(&$sandbox = NULL) {
  $spec = [
  'description' => 'A table to store a field.',
  'fields' => [
    'myfield1' => [
      'description' => 'Myfield1.',
      'type' => 'varchar',
      'length' => 255,
      'not null' => TRUE,
      'default' => '',
    ],
    'primary key' => ['myfield1'],
  ];
  $schema = Database::getConnection()->schema();
  $schema->createTable('mytable', $spec);
}
```

Try it!

- Create an update hook.

Dependencies

Dependencies

- Drupal can install other modules or include third party libraries automatically.

Dependencies

- Enforce Drupal module dependencies.

```yaml
name: My Module
type: module
description: 'My module'
core_version_requirement: ^8.8 || ^9

dependencies:
 - drupal:user
 - metatag:metatag
```

Dependencies

- Include library dependencies.

```
mymdoule.admin:
  version: VERSION
  css:
    theme:
      css/mymodule.admin.css: {}
  js:
    js/mymodule.admin.js: {}
  dependencies:
    - core/jquery
    - core/drupal
```

Default Configuration

Default Configuration

- Drupal can install configuration for you when you install the module.
- Useful for installing entity types or adding fields.
- Configuration files in **module/config/install** will be installed.
- Configuration files in **module/config/optional** will be installed if all their dependencies are met.

Default Configuration

```
config/
  install/
    mymodule.settings.yml
  optional/
    pathauto.pattern.mymodule.yml
```

- mymodule.settings.yml will be installed.
- pathauto.pattern.mymodule.yml will be installed if the Path Auto module is installed.

Coding Standards

Coding Standards

- Drupal has a number of coding standards covering PHP, JavaScript, YAML and CSS.
- Following them will make your module better, more secure, more maintainable and usable by third parties.

Coding Standards

```
phpcs --standard=Drupal,DrupalPractice
  --extensions=php,module,inc,install,test,profile,theme,css,info,txt,
  md,yml path/to/directory
```

Some Final Notes

Design Philosophy

- Think about modules in the most generic way possible. Even when naming it.
- Use contfiguration to control what your module acts upon.
- You should be thinking "this might make a good contrib module".
- Collaboration over competition.