laravel 4 - Cheat Sheet

Note: "<value>" : values in angle bracket are optional values.

Project on: https://github.com/Turaylon/laravel4-cheat-sheet

Composer

```
composer create-project laravel/laravel <folder>
```
Install laravel whit all the dependencies in a new <folder>. If no <folder> provided a laravel folder is created.

```
composer install <--dev>
```
Install all of the framework's dependencies. *--dev option* instal the require-dev additional dependency.

```
composer update <--no-dev>
```
Update all of the framework's dependencies. *--no-dev option* update whitout require-dev additional dependency. **Require-dev dependency are installed by default.**

```
composer dump-autoload
```
Update the autoloader. You should run it after adding a new class in your project.

Configuration

```
Config::get('file.key',<'default'>);
```
Accessing configuration value.

```
Config::set('key','value');
```
Set configuration value at runtime.

```
Config::has('key');
```
Determine if the given configuration value exists.

Routing

```
Route::get('uri/{var?}', function($var = null) );
```
Catch the GET uri and execute a closure. "?" after the var stands for optionl values in the uri.

```
Route::post('uri/{var}', function($var) );
```
Catch the POST uri and execute a closure. Optional values same as the get method.

```
Route::post('uri/{var}',array ('https' function() ));
```
Force the Route to be served over HTTPS

```
Route::post('uri/{var}', function($var) )
->where('var','[Regex]');
```
Rotue whit a regular expression constraints.

```
Route::filter('filter', function($route,$request,
$value) )
```
Defining a Route Filter. 'after' filters receive a $response as the third argument passed to the filter

```
Route::get('uri', array('before'=>'filter1:var|filter2'),
function() )
```
Attach filters to a route. Optional parameter can be passed.

```
Route::when('uri/*', 'filter');
```
Apply the routes on a pattern URI.

```
Route::get('uri', array('as'=>'name', function() ));
```
Naming a route. Usefull for referring to routes when generating redirects or URLs

```
Route::group(array('before'=>'filter', function() ));
```
Apply a set of filters to a group of routes. Routes are declared inside the closure.

Routing

```
Route::group(array('domain'=>'{var}.myapp.com'),
function() );
```
Handle sub-domains wildcard, and pass the wildcard to the routes inside the closure.

```
Route::group(array('prefix'=>'value'),
function() );
```
Prefix the routes in the closure whit the 'value' string.

```
Route::model('var'=>'Model');
Route::get('uri/{var}',function(Model $var) );
```
Bind a model 'Model' to the var 'var' and inject its instance into route.

```
Route::bind('var',function($value, $route){
    return Model::where('key', $value)->first();
});
```
Resolve the 'var' parameters whit a custom 'Model' data extraction.

Routing Controller

```
Route::get('uri', 'Controller@action');
```
Route te 'uri' to a specific 'action' of controller 'Controller'

```
Route::currentRouteAction();
```
Retrieve the name of the controller action being run.

```
Route::controller('uri', 'Controller');
```
Define a RESTful Controller to the 'uri'. Function in controller must be prefixed with the http verb.

```
Route::resource('uri', 'Controller');
```
Define a RESTful Controller that should manage a resource. Its a good practice build the controller with the command: **php artisan controller:make myController**

```
Route::get('uri', array( 'before' => 'filter',
        'uses'=> 'Controller@action'));
```
Filter a controller for 'uri'. Filters can also specified in the controller like: *$this->beforeFilter('filter');*

Input

```
Input::get('key',<'default'>);
```
Accessing input value.for all HTTP verb. 'GET' have priority on 'POST'.

```
Input::has('key');
```
Determining if an input value is present. Empty string are considered as no input value is present.

```
Input::all();
```
Getting all input for the request.

```
Input::only('key1', 'key2', 'keyN');
```
Getting only specified key of the request.

```
Input::except('key1', 'key2', 'keyN');
```
Getting all input request exept specified key.

```
Input::flash();
```
Save all the input in the session for the next request. You may easily chain input flashing onto a redirect in this way: *Redirect::to('uri')->withInput();*

Input

```
Input::flashOnly('key1','key2','keyN');
```
Save only specified input in the session for the next request.

```
Input::flashExcept('key1','key2','keyN');
```
Save all the input in the session except the key specified.

```
Input::old(<'key'>);
```
Retrieve the old input flashed in the session. If 'key' is specified, the input associated is returned otherwise all the input in session.

Files

```
Input::file('key');
```
Return an object that trpovides a variety of method for interacting whit the file.

```
Input::file('key')->move($destPath,<$filename>);
```
Move the uploaded file in a specific folder and eventually ranaming it.

```
Input::file('key')->getRealPath();
```
Retrieving the path to the uploaded file.

```
Input::file('key')->getSize();
```
Retrieving the size of the uploaded file.

```
Input::file('key')->getMimeType();
```
Retrieving the MIME Type of the uploaded file.

```
Input::hasFile('key');
```
Determine if 'key' file is uploaded.

Request

```
Request::path();
```
Get the Request URI.

```
Request::is('uri/*');
```
Determine if the request path matches a pattern.

```
Request::segment(1);
```
Get the specified uri segment.

```
Request::segments();
```
Get all uri segments.

```
Request::url();
```
Get the request URL.

```
Request::header('Content-Type');
```
Get the request Header.

```
Request::server('key');
```
Get $_SERVER['key'] value.

```
Request::ajax();
```
Determine if the Request is using AJAX.

```
Request::secure();
```
Determine if the Reqeust is over the HTTPS protocol.

Redirects

```
Redirect::to('uri');
```
Redirect to the specified 'uri'.

```
Redirect::route('routeName',<$params>);
```
Redirect to a named route. Parameters can be passed.

```
Redirect::action('Controller@action',<$params>);
```
Redirect to a controller action. Parameters can be passed.

```
Redirect::to('uri')->with('key','value');
```
Redirect with flash data. Array can be provided instead a couple of values.

Response

```
Response::make($content, $statusCode);
```
Create a custom responce. The object returned provides a variety of method for building HTTP responses.

```
Response::make($cont)->withCookie($cookie);
```
Attach a Cookie Object to the response. Cookie are generated with : *Cookie::make('name','value');*

```
Response::json($data);
```
Create a JSON response. A callback can be setted for JSONP response chaining *->setCallback($callback);* method.

```
Response::download($pathFile,$name,$headers);
```
Create a file download response. $name and $headers are optional

Views

```
Views::make('viewName', <$data>);
```
Parse a view 'viewName'. Optional $data can be passed to the view. Data can be passed alternatively chaining the method: *->with($data);*
Views can be stored in sub-folder and then use dots "." for directory separator Eg. "folder.view".

```
Views::share( 'key' , 'value');
```
Share the 'key' data to all the views.

```
Views::make( 'viewName')
->nest('child','child.view', <$data>);
```
Passing a Sub-view to a view.

```
Views::composer( 'viewName', function($view){
    $view->with($data);
});
```
Bind a callback to the 'viewName' views that pass $data every time the view is called.

Views - Controller layouts

```
//in class controller
protect $layout = 'layout.master';
//in the action of controller
$this->layout->content = Views::make('view');
```
Define a layout for the controller views.
The layout object will take the views that should be returned from actions.
There is no need to 'return' data from the action if is setted in the $this->layout->content.

Blade

```
{{ $var }}
{{ function() }} //return value is printed
{{{ $var }}} //escape the output
```
Echoing data.

```
<! -- app/views/layouts/master.blade.php -->
<html>
  <body>
    @section('sidebar')
      Here the default content.
    @show
    <div class="container">
      @yield('content')
    </div>
  </body>
</html>
```
Example of a master layout template.
@section are part whit default code that can be overwritten.
@yeld are part where content should be injected.

```
<! -- example of using a blade layout -->
@extends('layouts.master')

@section('sidebar')
  @parent
  <p>This is appended to the master sidebar.</p>
@stop

@section('content')
  <p>This is my body content.</p>
@stop
```
@extends defining a layout that should be extended
@section-@stop are part that overwrite the content
@parent include the parent @section content.

```
@if ( $statement == true )
  //do something
@elseif ( $statement == true )
  //do something
@else
  //do something else
@endif

@unless ( $statement == false)
  //do something
@endunless
```
If and unless statements.

```
@for ($i = 0; $i < 10; $i++)
  The current value is {{ $i }}
@endfor

@foreach ($users as $user)
  <p>This is user {{ $user->id }}</p>
@endforeach

@while ($statement == true)
  <p>I'm looping forever.</p>
@endwhile
```
Loops

```
@include( 'view.name' )
```
Include sub-views

```
@lang( 'language.line' )
@choice( 'language.line', $number )
```
Get language lines. @choice pick a singular or plural line based on $number.

```
{{-- this is a blade comment --}}
```
Comment that will not be rendered in HTML