

Laravel Cheat Sheet



hint: keep tapping the search, it scrolls to the result

Artisan

```
// Displays help for a given command
php artisan --help OR -h
// Do not output any message
php artisan --quiet OR -q
// Display this application version
php artisan --version OR -V
// Do not ask any interactive question
php artisan --no-interaction OR -n
// Force ANSI output
php artisan --ansi
// Disable ANSI output
php artisan --no-ansi
// The environment the command should run under
php artisan --env
// -v|vv|vvv Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
php artisan --verbose

// Display the framework change list
php artisan changes
// Remove the compiled class file
php artisan clear-compiled
// Put the application into maintenance mode
php artisan down
// Regenerate framework autoload files
php artisan dump-autoload
// Display the current framework environment
php artisan env
```

```
// Displays help for a command
php artisan help
// Lists commands
php artisan list

// Optimize the framework for better performance
php artisan optimize
// List all registered routes
php artisan routes
// Serve the application on the PHP development server
php artisan serve
// Interact with your application
php artisan tinker
// Bring the application out of maintenance mode
php artisan up
// Create a new package workbench
php artisan workbench

// Publish a package's assets to the public directory
php artisan asset:publish [--bench="vendor/package"] [--path="..."] [package]
// Create a migration for the password reminders table
php artisan auth:reminders-table
// Flush the application cache
php artisan cache:clear
// Create a new Artisan command (L3:task)
php artisan command:make name [--command="..."] [--path="..."] [--namespace="..."]
// Publish a package's configuration to the application
php artisan config:publish
// Create a new resourceful controller
php artisan controller:make [--bench="vendor/package"]
// Seed the database with records
php artisan db:seed [--class="..."] [--database="..."]
// Set the application key
php artisan key:generate

// Database migrations
```

```
php artisan migrate
php artisan migrate [--bench="vendor/package"] [--database="..."] [--path="..."] [--package="..."] [--pretend] [--seed]
// Create the migration repository
php artisan migrate:install [--database="..."]
// Create a new migration file
php artisan migrate:make name [--bench="vendor/package"] [--create] [--package="..."] [--path="..."] [--table="..."]
// Reset and re-run all migrations
php artisan migrate:refresh [--database="..."] [--seed]
// Rollback all database migrations
php artisan migrate:reset [--database="..."] [--pretend]
// Rollback the last database migration
php artisan migrate:rollback [--database="..."] [--pretend]

// Listen to a given queue
php artisan queue:listen [--queue="..."] [--delay="..."] [--memory="..."] [--timeout="..."] [connection]
// Subscribe a URL to an Iron.io push queue
php artisan queue:subscribe [--type="..."] queue url
// Process the next job on a queue
php artisan queue:work [--queue="..."] [--delay="..."] [--memory="..."] [--sleep] [connection]
// Create a migration for the session database table

php artisan session:table
// Publish a package's views to the application
php artisan view:publish [--path="..."] package
php artisan tail [--path="..."] [--lines="..."] [connection]
```

Composer

```
composer create-project laravel/laravel folder_name
composer install
composer update
composer dump-autoload [--optimize]
composer self-update
```

Routing □

```
Route::get('foo', function(){});  
Route::get('foo', 'ControllerName@function');  
Route::controller('foo', 'FooController');
```

RESTful Controllers

```
Route::resource('posts', 'PostsController');  
//Specify a subset of actions to handle on the route  
Route::resource('photo', 'PhotoController', ['only' => ['index', 'show']]);  
Route::resource('photo', 'PhotoController', ['except' => ['update', 'destroy']]);
```

Triggering Errors

```
App::abort(404);  
App::missing(function($exception){});  
throw new NotFoundHttpException;
```

Route Parameters

```
Route::get('foo/{bar}', function($bar){});  
Route::get('foo/{bar?}', function($bar = 'bar'){});
```

HTTP Verbs

```
Route::any('foo', function(){});  
Route::post('foo', function(){});  
Route::put('foo', function(){});  
Route::patch('foo', function(){});  
Route::delete('foo', function(){});  
// RESTful actions  
Route::resource('foo', 'FooController');
```

Secure Routes

```
Route::get('foo', array('https', function(){}));
```

Route Constraints

```
Route::get('foo/{bar}', function($bar){})
```

```
->where('bar', '[0-9]+');  
Route::get('foo/{bar}/{baz}', function($bar, $baz){})  
    ->where(array('bar' => '[0-9]+', 'baz' => '[A-Za-z]'))
```

```
// Set a pattern to be used across routes  
    Route::pattern('bar', '[0-9]+')
```

Filters

```
// Declare an auth filter  
Route::filter('auth', function(){});  
// Register a class as a filter  
Route::filter('foo', 'FooFilter');  
Route::get('foo', array('before' => 'auth', function(){}));  
// Routes in this group are guarded by the 'auth' filter  
Route::get('foo', array('before' => 'auth', function(){}));  
Route::group(array('before' => 'auth'), function(){});  
// Pattern filter  
Route::when('foo/*', 'foo');  
// HTTP verb pattern  
Route::when('foo/*', 'foo', array('post'));
```

Named Routes

```
Route::currentRouteName();  
Route::get('foo/bar', array('as' => 'foobar', function(){}));
```

Route Prefixing

```
// This route group will carry the prefix 'foo'  
Route::group(array('prefix' => 'foo'), function(){})
```

Route Namespacing

```
// This route group will carry the namespace 'Foo\Bar'  
Route::group(array('namespace' => 'Foo\Bar'), function(){})
```

Sub-Domain Routing

```
// {sub} will be passed to the closure  
Route::group(array('domain' => '{sub}.example.com'), function(){});
```

App

```
App::environment();  
// test equal to  
App::environment('local');  
App::runningInConsole();  
App::runningUnitTests();
```

Log

```
Log::info('info');  
Log::info('info',array('context'=>'additional info'));  
Log::error('error');  
Log::warning('warning');  
// get monolog instance  
Log::getMonolog();  
// add listener  
Log::listen(function($level, $message, $context) {});  
// get all ran queries.  
DB::getQueryLog();
```

URLs

```
URL::full();  
URL::current();  
URL::previous();  
URL::to('foo/bar', $parameters, $secure);  
URL::action('FooController@method', $parameters, $absolute);  
URL::route('foo', $parameters, $absolute);  
URL::secure('foo/bar', $parameters);  
URL::asset('css/foo.css', $secure);  
URL::secureAsset('css/foo.css');
```

```
URL::isValidUrl('http://example.com');
URL::getRequest();
URL::setRequest($request);
URL::getGenerator();
URL::setGenerator($generator);
```

Events

```
Event::fire('foo.bar', array($bar));
Event::listen('foo.bar', function($bar){});
Event::listen('foo.*', function($bar){});
Event::listen('foo.bar', 'FooHandler', 10);
Event::listen('foo.bar', 'BarHandler', 5);
Event::listen('foo.bar', function($event){ return false; });
Event::queue('foo', array($bar));
Event::flusher('foo', function($bar){});
Event::flush('foo');
Event::forget('foo');
Event::subscribe(new FooEventHandler);
```

Database

```
DB::connection('connection_name');
DB::statement('drop table users');
DB::listen(function($sql, $bindings, $time){ code_here; });
DB::transaction(function(){ transaction_code_here; });
// Cache a query for $time minutes
DB::table('users')->remember($time)->get();
// Escape raw input
DB::raw('sql expression here');
```

Selects

```
DB::table('name')->get();
DB::table('name')->distinct()->get();
DB::table('name')->select('column as column_alias')->get();
```

```
DB::table('name')->where('name', '=', 'John')->get();
DB::table('name')->whereBetween('column', array(1, 100))->get();
DB::table('name')->whereIn('column', array(1, 2, 3))->get();
DB::table('name')->whereNotIn('column', array(1, 2, 3))->get();
DB::table('name')->whereNull('column')->get();
DB::table('name')->whereNotNull('column')->get();
DB::table('name')->groupBy('column')->get();
// Default Eloquent sort is ascendant
DB::table('name')->orderBy('column')->get();
DB::table('name')->orderBy('column', 'desc')->get();
DB::table('name')->having('count', '>', 100)->get();
DB::table('name')->skip(10)->take(5)->get();
DB::table('name')->first();
DB::table('name')->pluck('column');
DB::table('name')->lists('column');
// Joins
DB::table('name')->join('table', 'name.id', '=', 'table.id')
    ->select('name.id', 'table.email');
```

Inserts, Updates, Deletes

```
DB::table('name')->insert(array('name' => 'John', 'email' => 'john@example.com'));
DB::table('name')->insertGetId(array('name' => 'John', 'email' => 'john@example.com'));
// Batch insert
DB::table('name')->insert(array(
    array('name' => 'John', 'email' => 'john@example.com')
    array('name' => 'James', 'email' => 'james@example.com')
));
// Update an entry
DB::table('name')->where('name', '=', 'John')
    ->update(array('email' => 'john@example2.com'));
// Delete everything from a table
DB::table('name')->delete();
// Delete specific records
DB::table('name')->where('id', '>', '10')->delete();
DB::table('name')->truncate();
```


Aggregates

```
DB::table('name')->count();
DB::table('name')->max('column');
DB::table('name')->min('column');
DB::table('name')->avg('column');
DB::table('name')->sum('column');
DB::table('name')->increment('column');
DB::table('name')->increment('column', $amount);
DB::table('name')->decrement('column');
DB::table('name')->decrement('column', $amount);
DB::table('name')->remember(5)->get();
DB::table('name')->remember(5, 'cache-key-name')->get();
DB::table('name')->cacheTags('my-key')->remember(5)->get();
DB::table('name')->cacheTags(array('my-first-key', 'my-second-key'))->remember(5)->get();
```

Raw Expressions

```
// return rows
DB::select('select * from users where id = ?', array('value'));
// return nr affected rows
DB::insert('insert into foo set bar=2');
DB::update('update foo set bar=2');
DB::delete('delete from bar');
// returns void
DB::statement('update foo set bar=2');
// raw expression inside a statement
DB::table('name')->select(DB::raw('count(*) as count, column2'))->get();
```

Eloquent

```
Model::create(array('key' => 'value'));
// Fill a model with an array of attributes, beware of mass assignment!
Model::fill($attributes);
Model::destroy(1);
Model::all();
Model::find(1);
```

```
// Find using dual primary key
Model::find(array('first', 'last'));
// Throw an exception if the lookup fails
Model::findOrFail(1);
// Find using dual primary key and throw exception if the lookup fails
Model::findOrFail(array('first', 'last'));
Model::where('foo', '=', 'bar')->get();
Model::where('foo', '=', 'bar')->first();
// dynamic
Model::whereFoo('bar')->first();
// Throw an exception if the lookup fails
Model::where('foo', '=', 'bar')->firstOrFail();
Model::where('foo', '=', 'bar')->count();
Model::where('foo', '=', 'bar')->delete();
//Output raw query
Model::where('foo', '=', 'bar')->toSql();
Model::whereRaw('foo = bar and cars = 2', array(20))->get();
Model::remember(5)->get();
Model::remember(5, 'cache-key-name')->get();
Model::cacheTags('my-tag')->remember(5)->get();
Model::cacheTags(array('my-first-key', 'my-second-key'))->remember(5)->get();
Model::on('connection-name')->find(1);
Model::with('relation')->get();
Model::all()->take(10);
Model::all()->skip(10);
```

Soft Delete

```
Model::withTrashed()->where('cars', 2)->get();
// Include the soft deleted models in the results
Model::withTrashed()->where('cars', 2)->restore();
Model::where('cars', 2)->forceDelete();
// Force the result set to only included soft deletes
Model::onlyTrashed()->where('cars', 2)->get();
```

Events

```
Model::creating(function($model){});
```

```
Model::created(function($model){});
Model::updating(function($model){});
Model::updated(function($model){});
Model::saving(function($model){});
Model::saved(function($model){});
Model::deleting(function($model){});
Model::deleted(function($model){});
Model::observe(new FooObserver);
```

Eloquent Configuration

```
// Disables mass assignment exceptions from being thrown from model inserts and updates
Eloquent::unguard();
// Renables any ability to throw mass assignment exceptions
Eloquent::reguard();
```

Schema

```
// Indicate that the table needs to be created
Schema::create('table', function($table)
{
    $table->increments('id');
});
// Specify a Connection
Schema::connection('foo')->create('table', function($table){});
// Rename the table to a given name
Schema::rename($from, $to);
// Indicate that the table should be dropped
Schema::drop('table');
// Indicate that the table should be dropped if it exists
Schema::dropIfExists('table');
// Determine if the given table exists
Schema::hasTable('table');
// Determine if the given table has a given column
Schema::hasColumn('table', 'column');
// Update an existing table
```

```
Schema::table('table', function($table){});  
// Indicate that the given columns should be renamed  
$table->renameColumn('from', 'to');  
// Indicate that the given columns should be dropped  
$table->dropColumn(string|array);  
// The storage engine that should be used for the table  
$table->engine = 'InnoDB';  
// Only work on MySQL  
$table->string('name')->after('email');
```

Indexes

```
$table->string('column')->unique();  
$table->primary('column');  
// Creates a dual primary key  
$table->primary(array('first', 'last'));  
$table->unique('column');  
$table->unique('column', 'key_name');  
// Creates a dual unique index  
$table->unique(array('first', 'last'));  
$table->unique(array('first', 'last'), 'key_name');  
$table->index('column');  
$table->index('column', 'key_name');  
// Creates a dual index  
$table->index(array('first', 'last'));  
$table->index(array('first', 'last'), 'key_name');  
$table->dropPrimary('table_column_primary');  
$table->dropUnique('table_column_unique');  
$table->dropIndex('table_column_index');
```

Foreign Keys

```
$table->foreign('user_id')->references('id')->on('users');  
$table->foreign('user_id')->references('id')->on('users')->onDelete('cascade'|'restrict'|'set null'|'no action');  
$table->foreign('user_id')->references('id')->on('users')->onUpdate('cascade'|'restrict'|'set null'|'no action');  
$table->dropForeign('posts_user_id_foreign');
```

Column Types

```
// Increments
$table->increments('id');
$table->bigIncrements('id');

// Numbers
$table->integer('votes');
$table->tinyInteger('votes');
$table->smallInteger('votes');
$table->mediumInteger('votes');
$table->bigInteger('votes');
$table->float('amount');
$table->double('column', 15, 8);
$table->decimal('amount', 5, 2);

//String and Text
$table->char('name', 4);
$table->string('email');
$table->string('name', 100);
$table->text('description');
$table->mediumText('description');
$table->longText('description');

//Date and Time
$table->date('created_at');
$table->dateTime('created_at');
$table->time('sunrise');
$table->timestamp('added_on');
$table->timestamps();
// Adds created_at and updated_at columns
$table->nullableTimestamps();

// Others
$table->binary('data');
$table->boolean('confirmed');
$table->softDeletes();
```

```
// Adds deleted_at column for soft deletes
$table->enum('choices', array('foo', 'bar'));
$table->rememberToken();
// Adds remember_token as VARCHAR(100) NULL
$table->morphs('parent');
// Adds INTEGER parent_id and STRING parent_type
->nullable()
->default($value)
->unsigned()
```

Input

```
Input::get('key');
// Default if the key is missing
Input::get('key', 'default');
Input::has('key');
Input::all();
// Only retrieve 'foo' and 'bar' when getting input
Input::only('foo', 'bar');
// Disregard 'foo' when getting input
Input::except('foo');
```

Session Input (flash)

```
// Flash input to the session
Input::flash();
// Flash only some of the input to the session
Input::flashOnly('foo', 'bar');
// Flash only some of the input to the session
Input::flashExcept('foo', 'baz');
// Retrieve an old input item
Input::old('key', 'default_value');
```

Files

```
// Use a file that's been uploaded
Input::file('filename');
```

```
// Determine if a file was uploaded
Input::hasFile('filename');
// Access file properties
Input::file('name')->getRealPath();
Input::file('name')->getClientOriginalName();
Input::file('name')->getClientOriginalExtension();
Input::file('name')->getSize();
Input::file('name')->getMimeType();
// Move an uploaded file
Input::file('name')->move($destinationPath);
// Move an uploaded file
Input::file('name')->move($destinationPath, $fileName);
```

Cache

```
Cache::put('key', 'value', $minutes);
Cache::add('key', 'value', $minutes);
Cache::forever('key', 'value');
Cache::remember('key', $minutes, function(){ return 'value' });
Cache::rememberForever('key', function(){ return 'value' });
Cache::forget('key');
Cache::has('key');
Cache::get('key');
Cache::get('key', 'default');
Cache::get('key', function(){ return 'default'; });
Cache::tags('my-tag')->put('key', 'value', $minutes);
Cache::tags('my-tag')->has('key');
Cache::tags('my-tag')->get('key');
Cache::tags('my-tag')->forget('key');
Cache::tags('my-tag')->flush();
Cache::increment('key');
Cache::increment('key', $amount);
Cache::decrement('key');
Cache::decrement('key', $amount);
Cache::section('group')->put('key', $value);
Cache::section('group')->get('key');
```

```
Cache::section('group')->flush();
```

Cookies

```
Cookie::get('key');  
Cookie::get('key', 'default');  
// Create a cookie that lasts for ever  
Cookie::forever('key', 'value');  
// Create a cookie that lasts N minutes  
Cookie::make('key', 'value', 'minutes');  
// Set a cookie before a response has been created  
Cookie::queue('key', 'value', 'minutes');  
// Forget cookie  
Cookie::forget('key');  
// Send a cookie with a response  
$response = Response::make('Hello World');  
// Add a cookie to the response  
$response->withCookie(Cookie::make('name', 'value', $minutes));
```

Sessions

```
Session::get('key');  
// Returns an item from the session  
Session::get('key', 'default');  
Session::get('key', function(){ return 'default'; });  
// Put a key / value pair in the session  
Session::put('key', 'value');  
// Push a value into an array in the session  
Session::push('foo.bar', 'value');  
// Returns all items from the session  
Session::all();  
// Checks if an item is defined  
Session::has('key');  
// Remove an item from the session  
Session::forget('key');
```



```
// Remove all of the items from the session
Session::flush();
// Generate a new session identifier
Session::regenerate();
// Flash a key / value pair to the session
Session::flash('key', 'value');
// Reflash all of the session flash data
Session::reflash();
// Reflash a subset of the current flash data
Session::keep(array('key1', 'key2'));
```

Requests

```
// url: http://xx.com/aa/bb
Request::url();
// path: /aa/bb
Request::path();
// getRequestUri: /aa/bb/?c=d
Request::getRequestUri();
// Returns user's IP
Request::getClientIp();
// getUri: http://xx.com/aa/bb/?c=d
Request::getUri();
// getQueryString: c=d
Request::getQueryString();
// Get the port scheme of the request (e.g., 80, 443, etc.)
Request::getPort();
// Determine if the current request URI matches a pattern
Request::is('foo/*');
// Get a segment from the URI (1 based index)
Request::segment(1);
// Retrieve a header from the request
Request::header('Content-Type');
// Retrieve a server variable from the request
Request::server('PATH_INFO');
// Determine if the request is the result of an AJAX call
```

```
Request::ajax();
// Determine if the request is over HTTPS
Request::secure();
// Get the request method
Request::method();
// Checks if the request method is of specified type
Request::isMethod('post');
// Get raw POST data
Request::instance()->getContent();
// Get requested response format
Request::format();
// true if HTTP Content-Type header contains */json
Request::isJson();
// true if HTTP Accept header is application/json
Request::wantsJson();
```

Responses

```
return Response::make($contents);
return Response::make($contents, 200);
return Response::json(array('key' => 'value'));
return Response::json(array('key' => 'value'))
    ->setCallback(Input::get('callback'));
return Response::download($filepath);
return Response::download($filepath, $filename, $headers);
// Create a response and modify a header value
$response = Response::make($contents, 200);
$response->header('Content-Type', 'application/json');
return $response;
// Attach a cookie to a response
return Response::make($content)
    ->withCookie(Cookie::make('key', 'value'));
```

Redirects

```
return Redirect::to('foo/bar');
return Redirect::to('foo/bar')->with('key', 'value');
return Redirect::to('foo/bar')->withInput(Input::get());
return Redirect::to('foo/bar')->withInput(Input::except('password'));
return Redirect::to('foo/bar')->withErrors($validator);
// Create a new redirect response to the previous location
return Redirect::back();
// Create a new redirect response to a named route
return Redirect::route('foobar');
return Redirect::route('foobar', array('value'));
return Redirect::route('foobar', array('key' => 'value'));
// Create a new redirect response to a controller action
return Redirect::action('FooController@index');
return Redirect::action('FooController@baz', array('value'));
return Redirect::action('FooController@baz', array('key' => 'value'));
// If intended redirect is not defined, defaults to foo/bar.
return Redirect::intended('foo/bar');
```

IoC ☐

```
App::bind('foo', function($app){ return new Foo; });
App::make('foo');
// If this class exists, it's returned
App::make('FooBar');
// Register a shared binding in the container
App::singleton('foo', function(){ return new Foo; });
// Register an existing instance as shared in the container
App::instance('foo', new Foo);
// Register a binding with the container
App::bind('FooRepositoryInterface', 'BarRepository');
// Register a service provider with the application
App::register('FooServiceProvider');
// Listen for object resolution
App::resolving(function($object){});
```

Security

Passwords

```
Hash::make('secretpassword');  
Hash::check('secretpassword', $hashedPassword);  
Hash::needsRehash($hashedPassword);
```

Auth

```
// Determine if the current user is authenticated  
Auth::check();  
// Get the currently authenticated user  
Auth::user();  
// Get the ID of the currently authenticated user  
Auth::id();  
// Attempt to authenticate a user using the given credentials  
Auth::attempt(array('email' => $email, 'password' => $password));  
// 'Remember me' by passing true to Auth::attempt()  
Auth::attempt($credentials, true);  
// Log in for a single request  
Auth::once($credentials);  
// Log a user into the application  
Auth::login(User::find(1));  
// Log the given user ID into the application  
Auth::loginUsingId(1);  
// Log the user out of the application  
Auth::logout();  
// Validate a user's credentials  
Auth::validate($credentials);  
// Attempt to authenticate using HTTP Basic Auth  
Auth::basic('username');  
// Perform a stateless HTTP Basic login attempt  
Auth::onceBasic();  
// Send a password reminder to a user  
Password::remind($credentials, function($message, $user){});
```

Encryption

```
Crypt::encrypt('secretstring');  
Crypt::decrypt($encryptedString);  
Crypt::setMode('ctr');  
Crypt::setCipher($cipher);
```

Mail

```
Mail::send('email.view', $data, function($message){});  
Mail::send(array('html.view', 'text.view'), $data, $callback);  
Mail::queue('email.view', $data, function($message){});  
Mail::queueOn('queue-name', 'email.view', $data, $callback);  
Mail::later(5, 'email.view', $data, function($message){});  
// Write all email to logs instead of sending  
Mail::pretend();
```

Messages

```
// These can be used on the $message instance passed into Mail::send() or Mail::queue()  
$message->from('email@example.com', 'Mr. Example');  
$message->sender('email@example.com', 'Mr. Example');  
$message->returnPath('email@example.com');  
$message->to('email@example.com', 'Mr. Example');  
$message->cc('email@example.com', 'Mr. Example');  
$message->bcc('email@example.com', 'Mr. Example');  
$message->replyTo('email@example.com', 'Mr. Example');  
$message->subject('Welcome to the Jungle');  
$message->priority(2);  
$message->attach('foo\bar.txt', $options);  
// This uses in-memory data as attachments  
$message->attachData('bar', 'Data Name', $options);  
// Embed a file in the message and get the CID  
$message->embed('foo\bar.txt');  
$message->embedData('foo', 'Data Name', $options);  
// Get the underlying Swift Message instance  
$message->getSwiftMessage();
```

Queues

```
Queue::push('SendMail', array('message' => $message));
Queue::push('SendEmail@send', array('message' => $message));
Queue::push(function($job) use $id {});
php artisan queue:listen
php artisan queue:listen connection
php artisan queue:listen --timeout=60
php artisan queue:work
```

Validation

```
Validator::make(
    array('key' => 'Foo'),
    array('key' => 'required|in:Foo')
);
Validator::extend('foo', function($attribute, $value, $params){});
Validator::extend('foo', 'FooValidator@validate');
Validator::resolver(function($translator, $data, $rules, $msgs)
{
    return new FooValidator($translator, $data, $rules, $msgs);
});
```

Rules

```
accepted
active_url
after:YYYY-MM-DD
before:YYYY-MM-DD
alpha
alpha_dash
alpha_num
array
between:1,10
confirmed
date
date_format:YYYY-MM-DD
```

```
different:fieldname
digits:value
digits_between:min,max
boolean
email
exists:table,column
image
in:foo,bar,...
not_in:foo,bar,...
integer
numeric
ip
max:value
min:value
mimes:jpeg,png
regex:[0-9]
required
required_if:field,value
required_with:foo,bar,...
required_with_all:foo,bar,...
required_without:foo,bar,...
required_without_all:foo,bar,...
same:field
size:value
timezone
unique:table,column,except,idColumn
url
```

Views

```
View::make('path/to/view');
View::make('foo/bar')->with('key', 'value');
View::make('foo/bar')->withKey('value');
View::make('foo/bar', array('key' => 'value'));
View::exists('foo/bar');
```

```
// Share a value across all views
View::share('key', 'value');
// Nesting views
View::make('foo/bar')->nest('name', 'foo/baz', $data);
// Register a view composer
View::composer('viewname', function($view){});
//Register multiple views to a composer
View::composer(array('view1', 'view2'), function($view){});
// Register a composer class
View::composer('viewname', 'FooComposer');
View::creator('viewname', function($view){});
```

Blade Templates ☐

```
@extends('layout.name')
// Begin a section
@section('name')
// End a section
@stop
// End a section and yield
@show
@parent
// Show a section in a template
@yield('name')
@include('view.name')
@include('view.name', array('key' => 'value'));
@lang('messages.name')
@choice('messages.name', 1);
@if
@else
@elseif
@endif
@unless
@endunless
@for
@endfor
```



```
@foreach
@endforeach
@while
@endwhile
// Echo content
{{ $var }}
// Echo escaped content
{{{ $var }}}
{{-- Blade Comment --}}
// Echoing Data After Checking For Existence
{{{ $name or 'Default' }}}
// Displaying Raw Text With Curly Braces
@{{ This will not be processed by Blade }}
```

Forms

```
Form::open(array('url' => 'foo/bar', 'method' => 'PUT'));
Form::open(array('route' => 'foo.bar'));
Form::open(array('route' => array('foo.bar', $parameter)));
Form::open(array('action' => 'FooController@method'));
Form::open(array('action' => array('FooController@method', $parameter)));
Form::open(array('url' => 'foo/bar', 'files' => true));
Form::close();
Form::token();
Form::model($foo, array('route' => array('foo.bar', $foo->bar)));
```

Form Elements

```
Form::label('id', 'Description');
Form::label('id', 'Description', array('class' => 'foo'));
Form::text('name');
Form::text('name', $value);
Form::text('name', $value, array('class' => 'name'));
Form::textarea('name');
Form::textarea('name', $value);
```

```
Form::textarea('name', $value, array('class' => 'name'));
Form::hidden('foo', $value);
Form::password('password');
Form::password('password', array('placeholder' => 'Password'));
Form::email('name', $value, array());
Form::file('name', array('class' => 'name'));
Form::checkbox('name', 'value');
// Generating a checkbox that is checked
Form::checkbox('name', 'value', true, array('class' => 'name'));
Form::radio('name', 'value');
// Generating a radio input that is selected
Form::radio('name', 'value', true, array('class' => 'name'));
Form::select('name', array('key' => 'value'));
Form::select('name', array('key' => 'value'), 'key', array('class' => 'name'));
Form::submit('Submit!', array('class' => 'name'));
Form::button('name', array('class' => 'name'));
Form::macro('fooField', function()
{
    return '<input type="custom"/>';
});
Form::fooField();
```

HTML Builder

```
HTML::macro('name', function(){});
// Convert an HTML string to entities
HTML::entities($value);
// Convert entities to HTML characters
HTML::decode($value);
// Generate a link to a JavaScript file
HTML::script($url, $attributes);
// Generate a link to a CSS file
HTML::style($url, $attributes);
// Generate an HTML image element
HTML::image($url, $alt, $attributes);
// Generate a HTML link
```

```
HTML::link($url, 'title', $attributes, $secure);
// Generate a HTTPS HTML link
HTML::secureLink($url, 'title', $attributes);
// Generate a HTML link to an asset
HTML::linkAsset($url, 'title', $attributes, $secure);
// Generate a HTTPS HTML link to an asset
HTML::linkSecureAsset($url, 'title', $attributes);
// Generate a HTML link to a named route
HTML::linkRoute($name, 'title', $parameters, $attributes);
// Generate a HTML link to a controller action
HTML::linkAction($action, 'title', $parameters, $attributes);
// Generate a HTML link to an email address
HTML::mailto($email, 'title', $attributes);
// Obfuscate an e-mail address to prevent spam-bots from sniffing it
HTML::email($email);
// Generate an ordered list of items
HTML::ol($list, $attributes);
// Generate an un-ordered list of items
HTML::ul($list, $attributes);
// Create a listing HTML element
HTML::listing($type, $list, $attributes);
// Create the HTML for a listing element
HTML::listingElement($key, $type, $value);
// Create the HTML for a nested listing attribute
HTML::nestedListing($key, $type, $value);
// Build an HTML attribute string from an array
HTML::attributes($attributes);
// Build a single attribute element
HTML::attributeElement($key, $value);
// Obfuscate a string to prevent spam-bots from sniffing it
HTML::obfuscate($value);
```

Strings

```
// Transliterate a UTF-8 value to ASCII
Str::ascii($value)
```

```
Str::camel($value)
Str::contains($haystack, $needle)
Str::endsWith($haystack, $needles)
// Cap a string with a single instance of a given value.
Str::finish($value, $cap)
Str::is($pattern, $value)
Str::length($value)
Str::limit($value, $limit = 100, $end = '...')
Str::lower($value)
Str::words($value, $words = 100, $end = '...')
Str::plural($value, $count = 2)
// Generate a more truly "random" alpha-numeric string.
Str::random($length = 16)
// Generate a "random" alpha-numeric string.
Str::quickRandom($length = 16)
Str::upper($value)
Str::title($value)
Str::singular($value)
Str::slug($title, $separator = '-')
Str::snake($value, $delimiter = '_')
Str::startsWith($haystack, $needles)
// Convert a value to studly caps case.
Str::studly($value)
Str::macro($name, $macro)
```

Localization

```
App::setLocale('en');
Lang::get('messages.welcome');
Lang::get('messages.welcome', array('foo' => 'Bar'));
Lang::has('messages.welcome');
Lang::choice('messages.apples', 10);
```

Files

```
File::exists('path');
File::get('path');
File::getRemote('path');
// Get a file's contents by requiring it
File::getRequire('path');
// Require the given file once
File::requireOnce('path');
// Write the contents of a file
File::put('path', 'contents');
// Append to a file
File::append('path', 'data');
// Delete the file at a given path
File::delete('path');
// Move a file to a new location
File::move('path', 'target');
// Copy a file to a new location
File::copy('path', 'target');
// Extract the file extension from a file path
File::extension('path');
// Get the file type of a given file
File::type('path');
// Get the file size of a given file
File::size('path');
// Get the file's last modification time
File::lastModified('path');
// Determine if the given path is a directory
File::isDirectory('directory');
// Determine if the given path is writable
File::isWritable('path');
// Determine if the given path is a file
File::isFile('file');
// Find path names matching a given pattern.
File::glob($patterns, $flag);
// Get an array of all files in a directory.
File::files('directory');
// Get all of the files from the given directory (recursive).
```

```
File::allFiles('directory');  
// Get all of the directories within a given directory.  
File::directories('directory');  
// Create a directory  
File::makeDirectory('path', $mode = 0777, $recursive = false);  
// Copy a directory from one location to another  
File::copyDirectory('directory', 'destination', $options = null);  
// Recursively delete a directory  
File::deleteDirectory('directory', $preserve = false);  
// Empty the specified directory of all files and folders  
File::cleanDirectory('directory');
```

Helpers ☐

Arrays

```
array_add($array, 'key', 'value');  
// Build a new array using a callback  
array_build($array, function(){});  
// Divide an array into two arrays. One with keys and the other with values  
array_divide($array);  
// Flatten a multi-dimensional associative array with dots  
array_dot($array);  
// Get all of the given array except for a specified array of items  
array_except($array, array('key'));  
// Fetch a flattened array of a nested array element  
array_fetch($array, 'key');  
// Return the first element in an array passing a given truth test  
array_first($array, function($key, $value){}, $default);  
// Strips keys from the array  
array_flatten($array);  
// Remove one or many array items from a given array using "dot" notation  
array_forget($array, 'foo');  
// Dot notation  
array_forget($array, 'foo.bar');  
// Get an item from an array using "dot" notation  
array_get($array, 'foo', 'default');
```

```
array_get($array, 'foo.bar', 'default');  
// Get a subset of the items from the given array  
array_only($array, array('key'));  
// Return array of key => values  
array_pluck($array, 'key');  
// Return and remove 'key' from array  
array_pull($array, 'key');  
// Set an array item to a given value using "dot" notation  
array_set($array, 'key', 'value');  
// Dot notation  
array_set($array, 'key.subkey', 'value');  
array_sort($array, function(){});  
// First element of an array  
head($array);  
// Last element of an array  
last($array);
```

Paths

```
app_path();  
// Get the path to the public folder  
public_path();  
// App root path  
base_path();  
// Get the path to the storage folder  
storage_path();
```

Strings

```
// Convert a value to camel case  
camel_case($value);  
// Get the class "basename" of the given object / class  
class_basename($class);  
// Escape a string  
e('<html>');  
// Determine if a given string starts with a given substring  
starts_with('Foo bar.', 'Foo');  
// Determine if a given string ends with a given substring
```

```
ends_with('Foo bar.', 'bar.');
```

// Convert a string to snake case

```
snake_case('fooBar');
```

// Determine if a given string contains a given substring

```
str_contains('Hello foo bar.', 'foo');
```

// Result: foo/bar/

```
str_finish('foo/bar', '/');
```

```
str_is('foo*', 'foobar');
```

```
str_plural('car');
```

```
str_random(25);
```

```
str_singular('cars');
```

// Result: FooBar

```
studly_case('foo_bar');
```

```
trans('foo.bar');
```

```
trans_choice('foo.bar', $count);
```

URLs and Links

```
action('FooController@method', $parameters);
```

```
link_to('foo/bar', $title, $attributes, $secure);
```

```
link_to_asset('img/foo.jpg', $title, $attributes, $secure);
```

```
link_to_route('route.name', $title, $parameters, $attributes);
```

```
link_to_action('FooController@method', $title, $params, $attrs);
```

// HTML Link

```
asset('img/photo.jpg', $title, $attributes);
```

// HTTPS link

```
secure_asset('img/photo.jpg', $title, $attributes);
```

```
secure_url('path', $parameters);
```

```
route($route, $parameters, $absolute = true);
```

```
url('path', $parameters = array(), $secure = null);
```

Miscellaneous

```
csrf_token();
```

```
dd($value);
```

```
value(function(){ return 'bar'; });
```

```
with(new Foo)->chainedMethod();
```


Unit testing

Install and run

```
// add to composer and update:  
"phpunit/phpunit": "4.0.*"  
// run tests (from project root)  
./vendor/bin/phpunit
```

Asserts

```
$this->assertTrue(true);  
$this->assertEquals('foo', $bar);  
$this->assertCount(1,$times);  
$this->assertResponseOk();  
$this->assertResponseStatus(403);  
$this->assertRedirectedTo('foo');  
$this->assertRedirectedToRoute('route.name');  
$this->assertRedirectedToAction('Controller@method');  
$this->assertViewHas('name');  
$this->assertViewHas('age', $value);  
$this->assertSessionHasErrors();  
// Asserting the session has errors for a given key...  
$this->assertSessionHasErrors('name');  
// Asserting the session has errors for several keys...  
$this->assertSessionHasErrors(array('name', 'age'));  
$this->assertHasOldInput();
```

Calling routes

```
$response = $this->call($method, $uri, $parameters, $files, $server, $content);  
$response = $this->callSecure('GET', 'foo/bar');  
$this->session(['foo' => 'bar']);  
$this->flushSession();  
$this->seed();  
$this->seed($connection);
```

SSH

Executing Commands

```
SSH::run(array $commands);  
SSH::into($remote)->run(array $commands); // specify remote, otherwise assumes default  
SSH::run(array $commands, function($line)  
{  
    echo $line.PHP_EOL;  
});
```

Tasks

```
SSH::define($taskName, array $commands); // define  
SSH::task($taskName, function($line) // execute  
{  
    echo $line.PHP_EOL;  
});
```

SFTP Uploads

```
SSH::put($localFile, $remotePath);  
SSH::putString($string, $remotePath);
```