









Artisan

Laravel 5.1 LTS Cheat Sheet

```
// Added in 5.1.11: http://laravel.com/docs/5.1/authorization#creating-policies
php artisan make:policy PostPolicy Laravel Forum
// Displays help for a given command Laravel 5.1 Docs
php artisan --help OR -h
// Do not output any message 
php artisan --quiet OR -q
// Display this application version by The EST Group
php artisan --version OR -V
// Do not ask any interactive question   
php artisan --no-interactive 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
// Remove the compiled class file
php artisan clear-compiled
// Display the current framework environment
php artisan env
// Displays help for a command
php artisan help
// Lists commands
php artisan list
// Interact with your application
php artisan tinker
// Put the application into maintenance mode
php artisan down
// Bring the application out of maintenance mode
php artisan up
// Optimize the framework for better performance
// --force Force the compiled class file to be written.
// --psr Do not optimize Composer dump-autoload.
php artisan optimize [--force] [--psr]
// Serve the application on the PHP development server
php artisan serve
// Change the default port
php artisan serve --port 8080
// Get it to work outside localhost
php artisan serve --host 0.0.0.0
// Set the application namespace
php artisan app:name namespace
```

```

// Flush expired password reset tokens
php artisan auth:clear-resets
// Flush the application cache
php artisan cache:clear
// Create a migration for the cache database table
php artisan cache:table
// Create a cache file for faster configuration loading
php artisan config:cache
// Remove the configuration cache file
php artisan config:clear
// In program
$exitCode = Artisan::call('config:cache');
// Seed the database with records
// --class      The class name of the root seeder (default: "DatabaseSeeder")
// --database   The database connection to seed
// --force      Force the operation to run when in production.
php artisan db:seed [--class="..."] [--database="..."] [--force]

// Generate the missing events and handlers based on registration
php artisan event:generate

// Create a new command handler class
// --command     The command class the handler handles.
php artisan handler:command [--command="..."] name
// Create a new event handler class
// --event       The event class the handler handles.
// --queued      Indicates the event handler should be queued.
php artisan handler:event [--event="..."] [--queued] name

// Set the application key
php artisan key:generate

// By default, this creates a self-handling command that isn't pushed to the queue.
// Pass this the --handler flag to generate a handler, and the --queued flag to make
it queued.
php artisan make:command [--handler] [--queued] name
// Create a new Artisan command
// --command     The terminal command that should be assigned. (default:
"command:name")
make:console [--command="..."] name
// Create a new resourceful controller
// --plain       Generate an empty controller class.
php artisan make:controller [--plain] name
php artisan make:controller App\Admin\Http\Controllers\DashboardController
// Create a new event class
php artisan make:event name
// Create a new middleware class
php artisan make:middleware name
// Create a new migration file
// --create      The table to be created.

```

```
// --table      The table to migrate.
php artisan make:migration [--create[="..."]] [--table[="..."]] name
// Create a new Eloquent model class
php artisan make:model name
// Create a new service provider class
php artisan make:provider name
// Create a new form request class
php artisan make:request name
// Database migrations
// --database    The database connection to use.
// --force       Force the operation to run when in production.
// --path        The path of migrations files to be executed.
// --pretend     Dump the SQL queries that would be run.
// --seed        Indicates if the seed task should be re-run.
php artisan migrate [--database[="..."]] [--force] [--path[="..."]] [--pretend] [--seed]
// Create the migration repository
php artisan migrate:install [--database[="..."]]
// Create a new migration file
// --seeder      The class name of the root seeder.
php artisan migrate:refresh [--database[="..."]] [--force] [--seed] [--seeder[="..."]]
// Rollback all database migrations
// --pretend     Dump the SQL queries that would be run.
php artisan migrate:reset [--database[="..."]] [--force] [--pretend]
// Rollback the last database migration
php artisan migrate:rollback [--database[="..."]] [--force] [--pretend]
// Show a list of migrations up/down
php artisan migrate:status
// Create a migration for the queue jobs database table
php artisan queue:table
// Listen to a given queue
// --queue       The queue to listen on
// --delay       Amount of time to delay failed jobs (default: 0)
// --memory      The memory limit in megabytes (default: 128)
// --timeout     Seconds a job may run before timing out (default: 60)
// --sleep       Seconds to wait before checking queue for jobs (default: 3)
// --tries       Number of times to attempt a job before logging it failed (default: 0)
php artisan queue:listen [--queue[="..."]] [--delay[="..."]] [--memory[="..."]] [--timeout[="..."]] [--sleep[="..."]] [--tries[="..."]] [connection]
// List all of the failed queue jobs
php artisan queue:failed
// Create a migration for the failed queue jobs database table
php artisan queue:failed-table
// Flush all of the failed queue jobs
php artisan queue:flush
// Delete a failed queue job
php artisan queue:forget
// Restart queue worker daemons after their current job
```

```

php artisan queue:restart
// Retry a failed queue job(id: The ID of the failed job)
php artisan queue:retry id
// Subscribe a URL to an Iron.io push queue
// queue: The name of Iron.io queue.
// url: The URL to be subscribed.
// --type          The push type for the queue.
php artisan queue:subscribe [--type[="..."]] queue url
// Process the next job on a queue
// --queue          The queue to listen on
// --daemon         Run the worker in daemon mode
// --delay          Amount of time to delay failed jobs (default: 0)
// --force          Force the worker to run even in maintenance mode
// --memory         The memory limit in megabytes (default: 128)
// --sleep          Number of seconds to sleep when no job is available (default: 3)
// --tries          Number of times to attempt a job before logging it failed (default:
0)
php artisan queue:work [--queue[="..."]] [--daemon] [--delay[="..."]] [--force] [--
memory[="..."]] [--sleep[="..."]] [--tries[="..."]] [connection]

// Create a route cache file for faster route registration
php artisan route:cache
// Remove the route cache file
php artisan route:clear
// List all registered routes
php artisan route:list

// Run the scheduled commands
php artisan schedule:run

// Create a migration for the session database table
php artisan session:table

// Publish any publishable assets from vendor packages
// --force          Overwrite any existing files.
// --provider       The service provider that has assets you want to publish.
// --tag            The tag that has assets you want to publish.
php artisan vendor:publish [--force] [--provider[="..."]] [--tag[="..."]]
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
composer require [options] [--] [vendor/packages]...

```

Config

```
Config::get('app.timezone');  
//get with Default value  
Config::get('app.timezone', 'UTC');  
//set Configuration  
Config::set('database.default', 'sqlite');
```

Route

```
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);  
$handler->missing(...) in ErrorServiceProvider::boot();  
throw new NotFoundException;
```

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');  
// Registering A Route For Multiple Verbs  
Route::match(['get', 'post'], '/', function() {});
```

Secure Routes(TBD)

```
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]+')
```

HTTP Middleware

```
// Assigning Middleware To Routes
Route::get('admin/profile', ['middleware' => 'auth', function(){}]);
Route::get('admin/profile', function(){}->middleware('auth');
```

Named Routes

```
Route::currentRouteName();

Route::get('foo/bar', array('as' => 'foobar', function(){}));
Route::get('user/profile', [
    'as' => 'profile', 'uses' => 'UserController@showProfile'
]);
Route::get('user/profile', 'UserController@showProfile')->name('profile');
$url = route('profile');
$redirect = redirect()->route('profile');
```

Route Prefixing

```
Route::group(['prefix' => 'admin'], function()
{
    Route::get('users', function(){
        return 'Matches The "/admin/users" URL';
    });
});
```

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(){});
```

Environment

```

$environment = app()->environment();
$environment = App::environment();
$environment = $app->environment();
// The environment is local
if ($app->environment('local')){}
// The environment is either local OR staging...
if ($app->environment('local', 'staging')){}

```

Log

```

// The logger provides the seven logging levels defined in RFC 5424:
// debug, info, notice, warning, error, critical, and alert.
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) {});

```

Query Logging

```

// enable the log
DB::connection()->enableQueryLog();
// get an array of the executed queries
DB::getQueryLog();

```

URL

```

URL::full();
URL::current();
URL::previous();
URL::to('foo/bar', $parameters, $secure);
URL::action('NewsController@item', ['id'=>123]);
// need be in appropriate namespace
URL::action('Auth\AuthController@logout');
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);

```

Event

```
Event::fire('foo.bar', array($bar));
// Register an event listener with the dispatcher.
// void listen(string|array $events, mixed $listener, int $priority)
Event::listen('App\Events\UserSignup', function($bar){});
Event::listen('foo.*', function($bar){});
Event::listen('foo.bar', 'FooHandler', 10);
Event::listen('foo.bar', 'BarHandler', 5);
// Stopping The Propagation Of An Event
// You may do so by returning false from your handler.
Event::listen('foo.bar', function($event){ return false; });
Event::subscribe('UserEventHandler');
```

DB

Basic Database Usage

```
DB::connection('connection_name');
// Running A Select Query
$results = DB::select('select * from users where id = ?', [1]);
$results = DB::select('select * from users where id = :id', ['id' => 1]);
// Running A General Statement
DB::statement('drop table users');
// Listening For Query Events
DB::listen(function($sql, $bindings, $time){ code_here; });
// Database Transactions
DB::transaction(function()
{
    DB::table('users')->update(['votes' => 1]);
    DB::table('posts')->delete();
});
DB::beginTransaction();
DB::rollBack();
DB::commit();
```

Query Builder

```
// Retrieving All Rows From A Table
DB::table('name')->get();
// Chunking Results From A Table
DB::table('users')->chunk(100, function($users)
{
    foreach ($users as $user)
    {

    }

});
// Retrieving A Single Row From A Table
```



```

$user = DB::table('users')->where('name', 'John')->first();
DB::table('name')->first();
// Retrieving A Single Column From A Row
$name = DB::table('users')->where('name', 'John')->pluck('name');
DB::table('name')->pluck('column');
// Retrieving A List Of Column Values
$roles = DB::table('roles')->lists('title');
$roles = DB::table('roles')->lists('title', 'name');
// Specifying A Select Clause
$users = DB::table('users')->select('name', 'email')->get();
$users = DB::table('users')->distinct()->get();
$users = DB::table('users')->select('name as user_name')->get();
// Adding A Select Clause To An Existing Query
$query = DB::table('users')->select('name');
$users = $query->addSelect('age')->get();
// Using Where Operators
$users = DB::table('users')->where('votes', '>', 100)->get();
$users = DB::table('users')
    ->where('votes', '>', 100)
    ->orWhere('name', 'John')
    ->get();
$users = DB::table('users')
    ->whereBetween('votes', [1, 100])->get();
$users = DB::table('users')
    ->whereNotBetween('votes', [1, 100])->get();
$users = DB::table('users')
    ->whereIn('id', [1, 2, 3])->get();
$users = DB::table('users')
    ->whereNotIn('id', [1, 2, 3])->get();
$users = DB::table('users')
    ->whereNull('updated_at')->get();
DB::table('name')->whereNotNull('column')->get();
// Dynamic Where Clauses
$admin = DB::table('users')->whereId(1)->first();
$johN = DB::table('users')
    ->whereIdAndEmail(2, 'john@doe.com')
    ->first();
$jane = DB::table('users')
    ->whereNameOrAge('Jane', 22)
    ->first();
// Order By, Group By, And Having
$users = DB::table('users')
    ->orderBy('name', 'desc')
    ->groupBy('count')
    ->having('count', '>', 100)
    ->get();
DB::table('name')->orderBy('column')->get();
DB::table('name')->orderBy('column', 'desc')->get();
DB::table('name')->having('count', '>', 100)->get();
// Offset & Limit

```

```
$users = DB::table('users')->skip(10)->take(5)->get();
```

Joins

```
// Basic Join Statement
DB::table('users')
    ->join('contacts', 'users.id', '=', 'contacts.user_id')
    ->join('orders', 'users.id', '=', 'orders.user_id')
    ->select('users.id', 'contacts.phone', 'orders.price')
    ->get();

// Left Join Statement
DB::table('users')
    ->leftJoin('posts', 'users.id', '=', 'posts.user_id')
    ->get();

// select * from users where name = 'John' or (votes > 100 and title <> 'Admin')
DB::table('users')
    ->where('name', '=', 'John')
    ->orWhere(function($query)
    {
        $query->where('votes', '>', 100)
            ->where('title', '<>', 'Admin');
    })
    ->get();
```

Aggregates

```
$users = DB::table('users')->count();
$price = DB::table('orders')->max('price');
$price = DB::table('orders')->min('price');
$price = DB::table('orders')->avg('price');
$total = DB::table('users')->sum('votes');
```

Raw Expressions

```
$users = DB::table('users')
    ->select(DB::raw('count(*) as user_count, status'))
    ->where('status', '<>', 1)
    ->groupBy('status')
    ->get();

// 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();
```

Inserts / Updates / Deletes / Unions / Pessimistic Locking

```
// Inserts
DB::table('users')->insert(
    ['email' => 'john@example.com', 'votes' => 0]
);
$id = DB::table('users')->insertGetId(
    ['email' => 'john@example.com', 'votes' => 0]
);
DB::table('users')->insert([
    ['email' => 'taylor@example.com', 'votes' => 0],
    ['email' => 'dayle@example.com', 'votes' => 0]
]);

// Updates
DB::table('users')
    ->where('id', 1)
    ->update(['votes' => 1]);
DB::table('users')->increment('votes');
DB::table('users')->increment('votes', 5);
DB::table('users')->decrement('votes');
DB::table('users')->decrement('votes', 5);
DB::table('users')->increment('votes', 1, ['name' => 'John']);

// Deletes
DB::table('users')->where('votes', '<', 100)->delete();
DB::table('users')->delete();
DB::table('users')->truncate();

// Unions
// The unionAll() method is also available, and has the same method signature as
// union.
$first = DB::table('users')->whereNull('first_name');
$users = DB::table('users')->whereNull('last_name')->union($first)->get();

// Pessimistic Locking
DB::table('users')->where('votes', '>', 100)->sharedLock()->get();
DB::table('users')->where('votes', '>', 100)->lockForUpdate()->get();
```

Model

Basic Usage

```
// Defining An Eloquent Model
class User extends Model {}
// generate Eloquent models
php artisan make:model User
// specify a custom table name
class User extends Model {
    protected $table = 'my_users';
}
```

More

```
Model::create(array('key' => 'value'));
```

```

// Find first matching record by attributes or create
Model::firstOrCreate(array('key' => 'value'));
// Find first record by attributes or instantiate
Model::firstOrCreate(array('key' => 'value'));
// Create or update a record matching attributes, and fill with values
Model::updateOrCreate(array('search_key' => 'search_value'), 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();
Model::where('foo', '=', 'bar')->exists();
// 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::on('connection-name')->find(1);
Model::with('relation')->get();
Model::all()->take(10);
Model::all()->skip(10);
// Default Eloquent sort is ascendant
Model::all()->orderBy('column');
Model::all()->orderBy('column', 'desc');

```

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();

```

Relationships

```

// One To One - User::phone()

```

```

return $this->hasOne('App\Phone', 'foreign_key', 'local_key');
// One To One - Phone::user(), The Inverse Of The Relation
return $this->belongsTo('App\User', 'foreign_key', 'other_key');

// One To Many - Post::comments()
return $this->hasMany('App\Comment', 'foreign_key', 'local_key');
// One To Many - Comment::post()
return $this->belongsTo('App\Post', 'foreign_key', 'other_key');

// Many To Many - User::roles();
return $this->belongsToMany('App\Role', 'user_roles', 'user_id', 'role_id');
// Many To Many - Role::users();
return $this->belongsToMany('App\User');
// Many To Many - Retrieving Intermediate Table Columns
$role->pivot->created_at;
// Many To Many - Pivot table with extra attributes
return $this->belongsToMany('App\Role')->withPivot('column1', 'column2');
// Many To Many - Automatically maintained created_at and updated_at timestamps
return $this->belongsToMany('App\Role')->withTimestamps();

// Has Many Through - Country::posts(), A Country model have
// many Post models through an intermediate User model (User::country_id)
return $this->hasManyThrough('App\Post', 'App\User', 'country_id', 'user_id');

// Polymorphic Relations - Photo::imageable()
return $this->morphTo();
// Polymorphic Relations - Staff::photos()
return $this->morphMany('App\Photo', 'imageable');
// Polymorphic Relations - Product::photos()
return $this->morphMany('App\Photo', 'imageable');
// Polymorphic Relations - Register the morphMap in your AppServiceProvider
Relation::morphMap([
    'Post' => App\Post::class,
    'Comment' => App\Comment::class,
]);

// Many To Many Polymorphic Relations - Tables: posts,videos,tags,taggables
// Post::tags()
return $this->morphToMany('App\Tag', 'taggable');
// Video::tags()
return $this->morphToMany('App\Tag', 'taggable');
// Tag::posts()
return $this->morphedByMany('App\Post', 'taggable');
// Tag::videos()
return $this->morphedByMany('App\Video', 'taggable');

// Querying Relations
$user->posts()->where('active', 1)->get();
// Retrieve all posts that have at least one comment...
$posts = App\Post::has('comments')->get();

```

```

// Retrieve all posts that have three or more comments...
$posts = Post::has('comments', '>=', 3)->get();
// Retrieve all posts that have at least one comment with votes...
$posts = Post::has('comments.votes')->get();
// Retrieve all posts with at least one comment containing words like foo%
$posts = Post::whereHas('comments', function ($query) {
    $query->where('content', 'like', 'foo%');
})->get();

// Eager Loading
$books = App\Book::with('author')->get();
$books = App\Book::with('author', 'publisher')->get();
$books = App\Book::with('author.contacts')->get();

// Lazy Eager Loading
$books->load('author', 'publisher');

// Inserting Related Models
$comment = new App\Comment(['message' => 'A new comment.']);
$post->comments()->save($comment);
// save multiple related models
$post->comments()->saveMany([
    new App\Comment(['message' => 'A new comment.']),
    new App\Comment(['message' => 'Another comment.']),
]);
$post->comments()->create(['message' => 'A new comment.']);

// Updating a belongsTo relationship
$user->account()->associate($account);
$user->save();
$user->account()->dissociate();
$user->save();

// Inserting Related Models - Many To Many Relationships
$user->roles()->attach($roleId);
$user->roles()->attach($roleId, ['expires' => $expires]);
// Detach a single role from the user...
$user->roles()->detach($roleId);
// Detach all roles from the user...
$user->roles()->detach();
$user->roles()->detach([1, 2, 3]);
$user->roles()->attach([1 => ['expires' => $expires], 2, 3]);

// Any IDs that are not in the given array will be removed from the intermediate
table.
$user->roles()->sync([1, 2, 3]);
// You may also pass additional intermediate table values with the IDs:
$user->roles()->sync([1 => ['expires' => true], 2, 3]);

```

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();
```

Pagination

```
// Auto-Magic Pagination
Model::paginate(15);
Model::where('cars', 2)->paginate(15);
// "Next" and "Previous" only
Model::where('cars', 2)->simplePaginate(15);
// Manual Paginator
Paginator::make($items, $totalItems, $perPage);
// Print page navigators in view
$variable->links();
```

Lang

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

File

```
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');
```

Unit**Test** 

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

```
// define
SSH::define($taskName, array $commands);
// execute
```

```
SSH::task($taskName, function($line)
{
    echo $line.PHP_EOL;
});
```

SFTP Uploads

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

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
```

```

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

```

Foreign Keys

```

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

```

Column Types

```

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

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

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

//Date and Time

```

```

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

// Others
$stable->binary('data');
$stable->boolean('confirmed');
// Adds deleted_at column for soft deletes
$stable->softDeletes();
$stable->enum('choices', array('foo', 'bar'));
// Adds remember_token as VARCHAR(100) NULL
$stable->rememberToken();
// Adds INTEGER parent_id and STRING parent_type
$stable->morphs('parent');
->nullable()
->default($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');
Input::flush();

```

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

```