

Laravel 11 Cheat Sheet

Color Legend: Blue = Essential/Daily Commands | Red = Advanced/Specialized Commands | Green = New in Laravel 11

Installation & Setup

<code>composer create-project laravel/laravel app-name</code>	Create new Laravel project
<code>composer global require laravel/installer</code>	Install Laravel installer globally
<code>laravel new app-name</code>	Create new Laravel project using installer
<code>laravel new app-name --git</code>	Create new project with git repo
<code>laravel new app-name --database=mysql</code>	Create project with specific database
<code>php artisan serve</code>	Start development server
<code>php artisan serve --host=0.0.0.0 --port=8080</code>	Start server with custom host/port
<code>php artisan --version</code>	Check Laravel version
<code>composer update</code>	Update Laravel dependencies
<code>composer install</code>	Install project dependencies
<code>composer install --no-dev --optimize-autoloader</code>	Production install
<code>npm install</code>	Install Node.js dependencies
<code>npm run dev</code>	Compile assets for development
<code>npm run build</code>	Compile assets for production
<code>npm run watch</code>	Watch and recompile assets

Artisan Commands

<code>php artisan list</code>	List all available commands
<code>php artisan help <command></code>	Get help for specific command
<code>php artisan make:model <name></code>	Create new model
<code>php artisan make:model <name> -m</code>	Create model with migration
<code>php artisan make:model <name> -mrc</code>	Create model, migration, resource controller
<code>php artisan make:model <name> -a</code>	Create model with all (migration, factory, seeder, policy, controller, form requests)
<code>php artisan make:controller <name></code>	Create new controller

<code>php artisan make:controller <name> --resource</code>	Create resource controller
<code>php artisan make:controller <name> --api</code>	Create API resource controller
<code>php artisan make:controller <name> --invokable</code>	Create single action controller
<code>php artisan make:migration <name></code>	Create new migration
<code>php artisan make:migration create_users_table</code>	Create migration with specific name
<code>php artisan make:migration add_column_to_table --table=users</code>	Add column to existing table
<code>php artisan make:seeder <name></code>	Create new seeder
<code>php artisan make:seeder UserSeeder</code>	Create specific seeder
<code>php artisan make:factory <name></code>	Create new factory
<code>php artisan make:factory UserFactory --model=User</code>	Create factory for specific model
<code>php artisan make:request <name></code>	Create new form request
<code>php artisan make:request StoreUserRequest</code>	Create specific form request
<code>php artisan make:middleware <name></code>	Create new middleware
<code>php artisan make:middleware CheckAge</code>	Create specific middleware
<code>php artisan make:policy <name></code>	Create new policy
<code>php artisan make:policy UserPolicy --model=User</code>	Create policy for specific model
<code>php artisan make:event <name></code>	Create new event
<code>php artisan make:listener <name></code>	Create new listener
<code>php artisan make:listener SendWelcomeEmail --event=UserRegistered</code>	Create listener for specific event
<code>php artisan make:job <name></code>	Create new job
<code>php artisan make:job ProcessPayment</code>	Create specific job
<code>php artisan make:mail <name></code>	Create new mail class
<code>php artisan make:mail WelcomeEmail --markdown=emails.welcome</code>	Create mail with markdown template
<code>php artisan make:notification <name></code>	Create new notification
<code>php artisan make:resource <name></code>	Create new API resource
<code>php artisan make:resource UserResource</code>	Create specific API resource
<code>php artisan make:test <name></code>	Create new test
<code>php artisan make:test UserTest --unit</code>	Create unit test
<code>php artisan make:test UserCanLoginTest --feature</code>	Create feature test
<code>php artisan make:command <name></code>	Create new artisan command

<code>php artisan make:provider <name></code>	Create new service provider
<code>php artisan make:rule <name></code>	Create new validation rule
<code>php artisan make:cast <name></code>	Create new custom cast
<code>php artisan make:component <name></code>	Create new Blade component
<code>php artisan make:observer <name></code>	Create new model observer

Routing

<code>Route::get('/uri', [Controller::class, 'method']);</code>	Basic GET route
<code>Route::post('/uri', [Controller::class, 'method']);</code>	POST route
<code>Route::put('/uri', [Controller::class, 'method']);</code>	PUT route
<code>Route::patch('/uri', [Controller::class, 'method']);</code>	PATCH route
<code>Route::delete('/uri', [Controller::class, 'method']);</code>	DELETE route
<code>Route::any('/uri', [Controller::class, 'method']);</code>	Route that responds to any HTTP verb
<code>Route::match(['get', 'post'], '/uri', [Controller::class, 'method']);</code>	Route responding to multiple verbs
<code>Route::resource('users', UserController::class);</code>	Resource route
<code>Route::apiResource('users', UserController::class);</code>	API resource route (no create/edit)
<code>Route::resource('users', UserController::class)->only(['index', 'show']);</code>	Partial resource routes
<code>Route::resource('users', UserController::class)->except(['destroy']);</code>	Resource routes except destroy
<code>Route::group(['prefix' => 'admin'], function () { ... });</code>	Route group with prefix
<code>Route::group(['middleware' => 'auth'], function () { ... });</code>	Route group with middleware
<code>Route::group(['namespace' => 'Admin'], function () { ... });</code>	Route group with namespace
<code>Route::middleware(['auth']->group(function () { ... });</code>	Route group with middleware
<code>Route::name('profile')->get('/profile', ...);</code>	Named route
<code>route('profile')</code>	Generate URL for named route
<code>route('profile', ['id' => 1])</code>	Generate URL with parameters
<code>Route::redirect('/here', '/there');</code>	Redirect route
<code>Route::redirect('/here', '/there', 301);</code>	Permanent redirect route
<code>Route::view('/welcome', 'welcome');</code>	Return view directly

<code>Route::view('/welcome', 'welcome', ['name' => 'Taylor']);</code>	Return view with data
<code>Route::fallback(function () { ... });</code>	Fallback route
<code>Route::domain('{account}.example.com')->group(...);</code>	Subdomain routing
<code>Route::where('id', '[0-9]+')->get('/user/{id}', ...);</code>	Route parameter constraints
<code>Route::whereNumber('id')->get('/user/{id}', ...);</code>	Numeric parameter constraint
<code>Route::whereAlpha('name')->get('/user/{name}', ...);</code>	Alphabetic parameter constraint
<code>Route::whereUuid('id')->get('/user/{id}', ...);</code>	UUID parameter constraint
<code>php artisan route:list</code>	List all routes
<code>php artisan route:list --name=user</code>	List routes with specific name
<code>php artisan route:list --method=GET</code>	List routes with specific method
<code>php artisan route:cache</code>	Cache routes for performance
<code>php artisan route:clear</code>	Clear route cache

Database & Migrations

<code>php artisan migrate</code>	Run pending migrations
<code>php artisan migrate --force</code>	Force run migrations in production
<code>php artisan migrate --pretend</code>	Show SQL that would be executed
<code>php artisan migrate --step</code>	Run migrations one by one
<code>php artisan migrate:rollback</code>	Rollback last migration
<code>php artisan migrate:rollback --step=5</code>	Rollback specific number of migrations
<code>php artisan migrate:reset</code>	Reset all migrations
<code>php artisan migrate:refresh</code>	Reset and re-run all migrations
<code>php artisan migrate:refresh --seed</code>	Reset, re-run migrations and seed
<code>php artisan migrate:fresh</code>	Drop all tables and re-run migrations
<code>php artisan migrate:fresh --seed</code>	Drop all tables, re-run migrations and seed
<code>php artisan migrate:status</code>	Show migration status
<code>php artisan make:migration create_users_table</code>	Create migration
<code>php artisan make:migration add_email_to_users_table --table=users</code>	Add column migration
<code>php artisan make:migration create_users_table --create=users</code>	Create table migration
<code>php artisan db:seed</code>	Run database seeders
<code>php artisan db:seed --class=UserSeeder</code>	Run specific seeder
<code>php artisan db:seed --force</code>	Force run seeders in production
<code>php artisan db:wipe</code>	Drop all tables, views, and types
<code>php artisan db:show</code>	Display information about database
<code>php artisan db:table users</code>	Display information about table
<code>php artisan db:monitor</code>	Monitor database connections
<code>php artisan tinker</code>	Interactive PHP shell
<code>php artisan schema:dump</code>	Dump current database schema
<code>php artisan schema:dump --prune</code>	Dump schema and prune migration files

Eloquent ORM

<code>User::all()</code>	Get all records
<code>User::find(\$id)</code>	Find record by ID

<code>User::findOrFail(\$id)</code>	Find record by ID or throw exception
<code>User::first()</code>	Get first record
<code>User::firstOrFail()</code>	Get first record or throw exception
<code>User::latest()->get()</code>	Get records ordered by latest
<code>User::oldest()->get()</code>	Get records ordered by oldest
<code>User::where('name', 'John')->get()</code>	Query with where clause
<code>User::where('age', '>', 18)->get()</code>	Query with comparison operator
<code>User::whereIn('id', [1, 2, 3])->get()</code>	Query with whereIn
<code>User::whereBetween('age', [18, 65])->get()</code>	Query with whereBetween
<code>User::whereNull('email_verified_at')->get()</code>	Query with whereNull
<code>User::whereNotNull('email_verified_at')->get()</code>	Query with whereNotNull
<code>User::whereDate('created_at', '2023-01-01')->get()</code>	Query by date
<code>User::whereYear('created_at', 2023)->get()</code>	Query by year
<code>User::whereMonth('created_at', 1)->get()</code>	Query by month
<code>User::select('name', 'email')->get()</code>	Select specific columns
<code>User::distinct()->get()</code>	Get distinct records
<code>User::orderBy('name', 'asc')->get()</code>	Order results ascending
<code>User::orderBy('created_at', 'desc')->get()</code>	Order results descending
<code>User::take(10)->get()</code>	Limit results
<code>User::skip(10)->take(10)->get()</code>	Skip and take (pagination)
<code>User::paginate(15)</code>	Paginate results
<code>User::simplePaginate(15)</code>	Simple pagination
<code>User::count()</code>	Count records
<code>User::max('age')</code>	Get maximum value
<code>User::min('age')</code>	Get minimum value
<code>User::avg('age')</code>	Get average value
<code>User::sum('salary')</code>	Get sum of values
<code>User::create(['name' => 'John', 'email' => '...'])</code>	Create new record
<code>User::insert([['name' => 'John'], ['name' => 'Jane']])</code>	Insert multiple records
<code>User::updateOrCreate(['email' => '...'], ['name' => 'John'])</code>	Update or create record

<code>User::firstOrCreate(['email' => '...'], ['name' => 'John'])</code>	Find or create record
<code>\$user->update(['name' => 'Jane'])</code>	Update record
<code>User::where('active', false)->update(['active' => true])</code>	Update multiple records
<code>\$user->delete()</code>	Delete record
<code>User::destroy([1, 2, 3])</code>	Delete multiple records by ID
<code>User::where('active', false)->delete()</code>	Delete multiple records by query
<code>User::with('posts')->get()</code>	Eager loading
<code>User::with(['posts', 'comments'])->get()</code>	Multiple eager loading
<code>User::with('posts:id,title,user_id')->get()</code>	Eager loading specific columns
<code>User::withCount('posts')->get()</code>	Eager loading with count

Blade Templates

<code>{{ \$variable }}</code>	Echo variable (escaped)
<code>{!! \$variable !!}</code>	Echo variable (unescaped)
<code>@if(\$condition) ... @endif</code>	Conditional statement
<code>@foreach(\$items as \$item) ... @endforeach</code>	Loop through items
<code>@extends('layout.app')</code>	Extend layout
<code>@section('content') ... @endsection</code>	Define section
<code>@yield('content')</code>	Yield section content
<code>@include('partials.header')</code>	Include partial view
<code>@auth ... @endauth</code>	Check if user is authenticated
<code>@guest ... @endguest</code>	Check if user is guest
<code>@csrf</code>	CSRF token field
<code>@method('PUT')</code>	Method spoofing field

Authentication

<code>php artisan make:auth</code>	Scaffold authentication views
<code>php artisan ui:auth</code>	Generate authentication scaffolding
<code>Auth::check()</code>	Check if user is authenticated
<code>Auth::user()</code>	Get authenticated user
<code>Auth::login(\$user)</code>	Log in user
<code>Auth::logout()</code>	Log out user
<code>auth()->user()</code>	Helper for authenticated user
<code>auth()->check()</code>	Helper to check authentication
<code>@auth ... @endauth</code>	Blade directive for auth check
<code>Route::middleware('auth')->group(...)</code>	Protect routes with auth

Middleware

<code>php artisan make:middleware CheckAge</code>	Create middleware
<code>Route::middleware('auth')->get(...)</code>	Apply middleware to route
<code>Route::middleware(['auth', 'admin'])->get(...)</code>	Multiple middleware
<code>protected \$middleware = [...] in Kernel.php</code>	Global middleware
<code>protected \$middlewareGroups = [...] in Kernel.php</code>	Middleware groups
<code>protected \$routeMiddleware = [...] in Kernel.php</code>	Route middleware
<code>\$request->user()</code>	Access user in middleware
<code>return \$next(\$request)</code>	Pass request to next middleware
<code>abort(403)</code>	Deny access in middleware

Validation Rules

<code>\$request->validate(['email' => 'required email'])</code>	Basic validation
<code>'required string max:255'</code>	Common string validation
<code>'required email unique:users'</code>	Email validation with unique
<code>'nullable integer min:1'</code>	Optional integer validation
<code>'required array min:1'</code>	Array validation
<code>'required file mimes:jpg,png max:2048'</code>	File validation
<code>'required date after:today'</code>	Date validation

<code>'required confirmed'</code>	Password confirmation
<code>'sometimes nullable string'</code>	Conditional validation
<code>Rule::exists('users', 'id')</code>	Database validation rule
<code>Rule::unique('users')->ignore(\$user->id)</code>	Unique with ignore
<code>'required regex:/^[A-Za-z]+\$/'</code>	Regex validation
<code>php artisan make:rule Uppercase</code>	Custom validation rule

API Development

<code>php artisan make:resource UserResource</code>	Create API resource
<code>php artisan make:resource UserCollection</code>	Create API collection
<code>return new UserResource(\$user)</code>	Return single resource
<code>return UserResource::collection(\$users)</code>	Return resource collection
<code>php artisan install:api</code>	Install Laravel Sanctum
<code>\$user->createToken('token-name')</code>	Create API token
<code>Route::middleware('auth:sanctum')->get(...)</code>	Protect API route
<code>return response()->json(\$data)</code>	Return JSON response
<code>return response()->json(\$data, 201)</code>	Return JSON with status
<code>abort_if(\$condition, 403)</code>	Conditional abort
<code>\$request->expectsJson()</code>	Check if request expects JSON

Queues & Jobs

<code>php artisan queue:work</code>	Start processing jobs
<code>php artisan queue:listen</code>	Listen for new jobs
<code>php artisan queue:restart</code>	Restart queue workers
<code>php artisan queue:failed</code>	List failed jobs
<code>php artisan queue:retry all</code>	Retry all failed jobs
<code>php artisan queue:retry 5</code>	Retry specific failed job
<code>php artisan queue:flush</code>	Delete all failed jobs
<code>php artisan queue:clear</code>	Delete all jobs from queue
<code>php artisan make:job ProcessPayment</code>	Create new job
<code>php artisan horizon</code>	Start Laravel Horizon dashboard

Cache Management

<code>Cache::put('key', 'value', 3600)</code>	Store cache item
<code>Cache::get('key')</code>	Get cache item
<code>Cache::remember('key', 3600, fn() => expensive_operation())</code>	Cache with fallback
<code>Cache::forget('key')</code>	Remove cache item
<code>Cache::flush()</code>	Clear all cache
<code>php artisan cache:clear</code>	Clear application cache
<code>php artisan cache:forget key</code>	Forget specific cache key
<code>Cache::tags(['people', 'artists'])->put('John', \$john, 60)</code>	Tagged cache
<code>Cache::lock('order-processing')->get(function () {...})</code>	Cache locks

Storage & Files

<code>Storage::disk('public')->put('file.txt', \$contents)</code>	Store file
<code>Storage::get('file.txt')</code>	Get file contents
<code>Storage::download('file.txt')</code>	Download file
<code>Storage::delete('file.txt')</code>	Delete file
<code>Storage::exists('file.txt')</code>	Check if file exists

<code>\$request->file('upload')->store('uploads')</code>	Store uploaded file
<code>php artisan storage:link</code>	Create storage symlink
<code>Storage::url('file.txt')</code>	Get file URL
<code>Storage::size('file.txt')</code>	Get file size
<code>Storage::lastModified('file.txt')</code>	Get last modified time

Laravel Sail (Docker)

<code>./vendor/bin/sail up</code>	Start all services
<code>./vendor/bin/sail up -d</code>	Start services in background
<code>./vendor/bin/sail down</code>	Stop all services
<code>./vendor/bin/sail artisan migrate</code>	Run migrations in container
<code>./vendor/bin/sail composer install</code>	Install dependencies in container
<code>./vendor/bin/sail npm run dev</code>	Run npm in container
<code>./vendor/bin/sail test</code>	Run tests in container
<code>./vendor/bin/sail shell</code>	Access container shell
<code>./vendor/bin/sail mysql</code>	Access MySQL in container
<code>./vendor/bin/sail redis</code>	Access Redis in container

Testing

<code>php artisan make:test UserTest</code>	Create test class
<code>php artisan test</code>	Run all tests
<code>php artisan test --filter=UserTest</code>	Run specific test
<code>\$this->assertEquals(\$expected, \$actual)</code>	Assert equality
<code>\$this->assertTrue(\$condition)</code>	Assert true
<code>\$this->assertDatabaseHas('users', [...])</code>	Assert database record exists
<code>\$this->get('/users')</code>	Make GET request in test
<code>\$this->post('/users', \$data)</code>	Make POST request in test
<code>\$this->actingAs(\$user)</code>	Authenticate user in test
<code>\$this->assertRedirect('/dashboard')</code>	Assert redirect

Inertia.js Integration

<code>composer require inertiajs/inertia-laravel</code>	Install Inertia.js Laravel adapter
<code>php artisan inertia:middleware</code>	Create Inertia middleware
<code>npm install @inertiajs/vue3</code>	Install Inertia Vue 3 adapter
<code>npm install @inertiajs/react</code>	Install Inertia React adapter
<code>Inertia::render('Users/Index', ['users' => \$users])</code>	Render Inertia page with data
<code>return inertia('Users/Show', compact('user'))</code>	Return Inertia response (helper)
<code>Inertia::location('/dashboard')</code>	Redirect with Inertia
<code>\$request->header('X-Inertia')</code>	Check if request is from Inertia
<code>Inertia::share('auth.user', fn() => auth()->user())</code>	Share data globally
<code>Inertia::version(fn() => md5_file(public_path('mix-manifest.json')))</code>	Asset versioning
<code><Head title='Page Title' /></code>	Set page title (Vue/React)
<code>\$page.props.user</code>	Access shared props (Vue/React)
<code>import { Link } from '@inertiajs/vue3'</code>	Inertia Link component (Vue)
<code>import { router } from '@inertiajs/vue3'</code>	Inertia router (Vue)
<code>router.visit('/users')</code>	Programmatic navigation
<code>router.post('/users', form)</code>	POST request with Inertia
<code>\$page.props.errors</code>	Access validation errors

Livewire Components

<code>composer require livewire/livewire</code>	Install Livewire
<code>php artisan make:livewire Counter</code>	Create Livewire component
<code>php artisan make:livewire Users/Index</code>	Create nested Livewire component
<code><livewire:counter /></code>	Render Livewire component
<code>@livewire('counter')</code>	Render with Blade directive
<code>public \$count = 0;</code>	Define public property
<code>public function increment() { \$this->count++; }</code>	Define action method
<code>wire:click='increment'</code>	Wire click event
<code>wire:model='name'</code>	Two-way data binding
<code>wire:submit.prevent='save'</code>	Wire form submission

<code>\$this->validate(['name' => 'required']);</code>	Validate in Livewire
<code>\$this->emit('userSaved');</code>	Emit event
<code>protected \$listeners = ['userSaved' => 'refreshUsers'];</code>	Listen to events
<code>wire:loading</code>	Show loading state
<code>wire:offline</code>	Show offline state
<code>\$this->skipRender();</code>	Skip component re-render

Deployment & Optimization

<code>php artisan config:cache</code>	Cache configuration
<code>php artisan route:cache</code>	Cache routes
<code>php artisan view:cache</code>	Cache views
<code>php artisan config:clear</code>	Clear config cache
<code>php artisan route:clear</code>	Clear route cache
<code>php artisan view:clear</code>	Clear view cache
<code>php artisan cache:clear</code>	Clear application cache
<code>composer install --optimize-autoloader --no-dev</code>	Optimize for production
<code>php artisan migrate --force</code>	Run migrations in production
<code>npm run build</code>	Build assets for production

■ Laravel 11 Tips & Best Practices:

- Use Eloquent relationships and eager loading for performance
- Always validate user input using Form Requests
- Leverage middleware for cross-cutting concerns (auth, CORS, etc.)
- Use Laravel Sanctum for API authentication
- Implement proper error handling with custom exception classes
- Use database seeders and factories for testing data
- Write comprehensive tests (Feature + Unit tests)
- Use Laravel's built-in caching mechanisms (Redis recommended)
- Follow PSR standards and use Laravel Pint for code formatting
- Use environment variables for all configuration
- Implement proper database indexing and query optimization
- Use Laravel's queue system for background jobs
- Use Laravel Horizon for queue monitoring in production
- Leverage Laravel Sail for consistent development environments

■ Laravel 11 New Features:

- Improved artisan commands with better UX
- Enhanced API resource handling
- Better Docker integration with Sail
- Improved testing capabilities
- Enhanced security features

■ Essential Packages for Laravel 11:

- Laravel Debugbar ([barryvdh/laravel-debugbar](#))
- Laravel IDE Helper ([barryvdh/laravel-ide-helper](#))
- Laravel Telescope ([laravel/telescope](#))
- Laravel Horizon ([laravel/horizon](#))
- Laravel Sanctum (built-in API authentication)
- Laravel Pint (built-in code formatting)
- Spatie Laravel packages (permissions, media, etc.)
- Laravel Livewire for reactive components
- Inertia.js for modern SPA development