Popular Artisan Commands in Laravel

Generated on May 08, 2025

Introduction

Laravel's Artisan is a powerful command-line tool that streamlines development by providing commands for generating code, managing databases, debugging, and optimizing applications. This document outlines the most commonly used Artisan commands, grouped by purpose, with examples and use cases.

1 General Commands

1.1 Listing Available Commands

`php artisan list`
Purpose: Displays all available Artisan commands with descriptions.
Use Case: Explore or recall available commands.
Example:

```
php artisan list
```

1.2 Getting Command Help

`php artisan help [command]`
Purpose: Shows detailed help for a specific command, including options.
Use Case: Understand a command's usage.
Example:

```
php artisan help make:model
```

1.3 Checking Laravel Version

`php artisan -version`
Purpose: Displays the current Laravel version.
Use Case: Verify the installed version.
Example:

```
php artisan --version
```

2 Generating Files (Make Commands)

2.1 Creating a Model

`php artisan make:model [Name]`
Purpose: Generates an Eloquent model class.
Options:

- `-m`: Creates a migration.
- `-c`: Creates a controller.
- `-r`: Generates a resourceful controller.
- `-f`: Creates a factory.

- `-all`: Generates all related files.

Use Case: Create a model for a database table.
Example:

```
php artisan make:model Post -mcr
```

2.2 Creating a Controller

`php artisan make:controller [Name]`
Purpose: Generates a controller class.
Options:

- `-resource`: Creates a resourceful controller.
- `-api`: Creates an API controller.

Use Case: Handle HTTP requests.
Example:

```
php artisan make:controller PostController --resource
```

2.3 Creating a Migration

`php artisan make:migration [Name]`
Purpose: Creates a database migration file.
Use Case: Define schema changes.
Example:

```
php artisan make:migration create_posts_table
```

2.4 Creating a Seeder

`php artisan make:seeder [Name]`
Purpose: Creates a database seeder class.
Use Case: Populate the database with data.
Example:

```
php artisan make:seeder UsersTableSeeder
```

2.5 Creating a Factory

`php artisan make:factory [Name]`
Purpose: Creates a model factory for fake data.
Use Case: Generate test data.
Example:

```
php artisan make:factory PostFactory
```

2.6 Creating a Form Request

`php artisan make:request [Name]`
Purpose: Creates a form request class for validation.
Use Case: Validate HTTP requests.
Example:

```
php artisan make:request StorePostRequest
```

2.7 Creating a Middleware

`php artisan make:middleware [Name]`
Purpose: Creates a middleware class.
Use Case: Filter or modify requests.
Example:

```
php artisan make:middleware CheckRole
```

2.8 Creating a Resource

`php artisan make:resource [Name]`
Purpose: Creates a resource class for JSON responses.
Use Case: Format API responses.
Example:

```
php artisan make:resource PostResource
```

3 Database Commands

3.1 Running Migrations

`php artisan migrate`
Purpose: Applies pending migrations to update the schema.
Use Case: Update the database structure.
Example:

```
php artisan migrate
```

3.2 Rolling Back Migrations

`php artisan migrate:rollback`
Purpose: Reverts the last batch of migrations.
Use Case: Undo recent schema changes.
Example:

```
php artisan migrate:rollback
```

3.3 Refreshing Migrations

`php artisan migrate:refresh`
Purpose: Rolls back and re-runs all migrations.
Use Case: Reset and reapply schema.
Example:

```
php artisan migrate:refresh
```

3.4 Dropping and Re-running Migrations

`php artisan migrate:fresh`
Purpose: Drops all tables and re-runs migrations.
Use Case: Start with a clean database.
Example:

```
php artisan migrate:fresh
```

3.5 Seeding the Database

`php artisan db:seed`
Purpose: Runs seeders to populate the database.
Options:

- `--class`: Specifies a seeder class.

Use Case: Insert test or initial data.
Example:

```
php artisan db:seed --class=UsersTableSeeder
```

3.6 Wiping the Database

`php artisan db:wipe`
Purpose: Drops all tables and data.
Use Case: Clear the database.
Example:

```
php artisan db:wipe
```

4 Development and Debugging

4.1 Starting a Development Server

`php artisan serve`
Purpose: Starts a local server (default: `http://localhost:8000`).
Use Case: Test the application locally.
Example:

```
php artisan serve --port=8080
```

4.2 Interactive REPL

`php artisan tinker`
Purpose: Opens an interactive PHP REPL.
Use Case: Test Eloquent queries or Laravel components.
Example:

```
php artisan tinker
```

4.3 Listing Routes

`php artisan route:list`
Purpose: Displays all registered routes.
Options:

- `-method`: Filters by HTTP method.
- `-path`: Filters by path.

Use Case: Debug routing.
Example:

```
php artisan route:list --method=GET
```

4.4 Clearing Cache

`php artisan cache:clear`
Purpose: Clears the application cache.
Use Case: Resolve issues with stale cache.
Example:

```
php artisan cache:clear
```

4.5 Caching Configuration

`php artisan config:cache`
Purpose: Caches configuration files.
Use Case: Optimize performance.
Example:

```
php artisan config:cache
```

4.6 Clearing Views

`php artisan view:clear`
Purpose: Clears cached views.
Use Case: Ensure updated views are rendered.
Example:

```
php artisan view:clear
```

5 Queue and Job Commands

5.1 Processing Queued Jobs

`php artisan queue:work`
Purpose: Starts a queue worker to process jobs.
Use Case: Handle background tasks.
Example:

```
php artisan queue:work
```

5.2 Listening for Jobs

`php artisan queue:listen`
Purpose: Listens for and processes queued jobs in real-time.
Use Case: Process jobs during development.
Example:

```
php artisan queue:listen
```

5.3 Creating a Job

`php artisan make:job [Name]`
Purpose: Creates a queueable job class.
Use Case: Define background tasks.
Example:

```
php artisan make:job SendEmailJob
```

6 Testing Commands

6.1 Running Tests

`php artisan test`
Purpose: Runs the test suite (PHPUnit or Pest).
Use Case: Verify application functionality.
Example:

```
php artisan test
```

6.2 Creating a Test

`php artisan make:test [Name]`
Purpose: Creates a test class.
Options:

- `-unit`: Creates a unit test.
- `-pest`: Creates a Pest test.

Use Case: Write unit or feature tests.
Example:

```
php artisan make:test UserTest --unit
```

7 Optimization Commands

7.1 Optimizing the Application

`php artisan optimize`
Purpose: Caches bootstrap files for performance.
Use Case: Prepare for production.
Example:

```
php artisan optimize
```

7.2 Caching Routes

`php artisan route:cache`
Purpose: Caches routes for faster resolution.
Use Case: Improve performance.
Example:

```
php artisan route:cache
```

7.3 Clearing Route Cache

`php artisan route:clear`
Purpose: Clears the route cache.
Use Case: Refresh routes after changes.
Example:

```
php artisan route:clear
```

8 Authentication Commands

8.1 Installing Breeze

`php artisan breeze:install`
Purpose: Installs Laravel Breeze for minimal authentication.
Use Case: Set up authentication quickly.
Example:

```
php artisan breeze:install
```

8.2 Installing Jetstream

`php artisan jetstream:install`
Purpose: Installs Laravel Jetstream for advanced authentication.
Options:

- `-livewire`: Uses Livewire.
- `-inertia`: Uses Inertia.js.

Use Case: Set up authentication with team features.
Example:

```
php artisan jetstream:install inertia
```

9 Miscellaneous Commands

9.1 Generating Application Key

`php artisan key:generate`
Purpose: Generates a new application key.
Use Case: Secure a new or cloned project.
Example:

```
php artisan key:generate
```

9.2 Linking Storage

`php artisan storage:link`
Purpose: Creates a symbolic link for public storage.
Use Case: Make storage files accessible.
Example:

```
php artisan storage:link
```

9.3 Creating a Custom Command

`php artisan make:command [Name]`
Purpose: Creates a custom Artisan command.
Use Case: Define custom CLI tasks.
Example:

```
php artisan make:command SendDailyReport
```

10 Tips for Using Artisan

- **Custom Commands**: Use `make:command` to create custom CLI tasks.
- **Options**: Explore command options with `php artisan help [command]`.
- **Chaining**: Combine commands (e.g., `migrate:fresh -seed`).
- **Environment**: Ensure the `.env` file is configured correctly.