



Desarrollo de aplicaciones con Symfony 4

Commands



Commands

El framework Symfony proporciona muchos comandos a través del script bin / console (por ejemplo, el conocido bin / console cache: clear command). Estos comandos se crean con el componente de la Consola. También puedes usarlo para crear tus propios comandos.

Los comandos de Symfony deben estar registrados como servicios y etiquetados con la etiqueta console.command. Si estamos utilizando el autowire y autoconfigure no es necesario hacer nada.

Commands: Creando un command

Los comandos se definen en las clases que extienden de Command.

```
// src/Command/IssueCommand.php
namespace App\Command;
use Symfony\Component\Console\Command\Command;
class IssueCommand extends Command
{
    protected static $defaultName = 'app:issue';
    protected function configure(){...}
    protected function execute(InputInterface $input, OutputInterface $output) {...}
}
```

Commands: Creando un command

Lo primero que deberemos hacer es darle un nombre

```
protected static $defaultName = 'app:issue';
```

Y opcionalmente darle una configuración extra en el método configure:

```
protected function configure()  
{  
    $this  
        ->setDescription('Comando de prueba.')  
        // si lo ejecutas con "--help"  
        ->setHelp('Este comando es una prueba.')  
    ;  
}
```

Commands: Creando un command

Pasando parámetros

```
protected function configure()
{
    $this
        ->addArgument('name', InputArgument::REQUIRED, '¿cual es tu nombre?')
        ->addArgument('last_name', InputArgument::OPTIONAL, '¿y apellido?')
    ;
}
```

Para recuperarlo (sería dentro del método execute que veremos a continuación):

```
$name = $input->getArgument('name');
```

Commands: Creando un command

Podemos ver todas la opciones de configuración en:

<https://symfony.com/doc/current/console/input.html>

Commands: Creando un command

El método `configure()` se llama automáticamente al final del `__construct` del `command`. Si su comando define su propio constructor, establezca las propiedades primero y luego llame al constructor principal, para que esas propiedades estén disponibles en el método `configure()`:

```
public function __construct($param)
{
    $this->param = $param;

    parent::__construct();
}
```

Commands

Y finalmente pondríamos la lógica de nuestro command dentro del método execute:

```
protected function execute(InputInterface $input, OutputInterface
$output)
{
    $output->writeln([
        'Hola',
        '=====',
        '',
    ]);
}
```


Commands: Mostrando mensajes por pantalla

// outputs multiple lines to the console (adding "\n" at the end of each line)

```
$output->writeln([  
    'User Creator',  
    '=====  
]);
```

// the value returned by someMethod() can be an iterator

// that generates and returns the messages with the 'yield' PHP keyword

```
$output->writeln($this->someMethod());
```

// outputs a message followed by a "\n"

```
$output->writeln('Whoa!');
```

// outputs a message without adding a "\n" at the end of the line

```
$output->write('You are about to ');
```

```
$output->write('create a user.');
```

Commands: Ejecutando el command

Si recordamos, lo primero que hemos hecho ha sido darle un nombre:

```
protected static $defaultName = 'app:issue';
```

Para ejecutarlo:

```
$ php bin/console app:issue
```

Commands: Inyectando servicios

De la misma forma que en el resto de nuestro servicios, al fin y al cabo, es un servicio más de nuestra aplicación:

```
private $issueManager;
```

```
public function __construct(IssueManager $issueManager)
{
    $this->issueManager = $issueManager;

    parent::__construct();
}
```

Commands: Ciclo de vida

Los comandos tienen tres métodos que se ejecutan en un determinado orden cada vez el comando se ejecuta:

initialize () (opcional)

Este método se ejecuta antes que los métodos `interact ()` y `execute ()`. Su propósito principal es inicializar las variables utilizadas en el resto de los métodos de comando.

Commands: Ciclo de vida

interact () (opcional)

Este método se ejecuta después de `initialize ()` y antes de `execute ()`. Su propósito es verificar si faltan algunas de las opciones / argumentos y solicitar de forma interactiva al usuario esos valores. Este es el último lugar donde puede solicitar opciones / argumentos que faltan. Después de este comando, las opciones / argumentos que faltan darán como resultado un error.

execute () (requerido)

Este método se ejecuta después de `interact ()` e `inicializar ()`. Contiene la lógica que desea que ejecute el comando.

Commands: Pedir parámetros por pantalla.

Pedir confirmación

```
use Symfony\Component\Console\Question\Question;
// ...
public function execute(InputInterface $input, OutputInterface $output) {
    $helper = $this->getHelper('question');
    $question = new ConfirmationQuestion('Continue with this action?', false);

    if (!$helper->ask($input, $output, $question)) {
        return;
    }
}
```

Commands: Pedir parámetros por pantalla.

Pedir información

```
use Symfony\Component\Console\Question\Question;
// ...
public function execute(InputInterface $input, OutputInterface $output) {
    // ...
    $question = new Question('Please enter the name of the bundle',
    'AcmeDemoBundle');

    $bundleName = $helper->ask($input, $output, $question);
}
```

Commands: Pedir parámetros por pantalla.

Choice:

```
use Symfony\Component\Console\Question\ChoiceQuestion;
// ...
$helper = $this->getHelper('question');
$question = new ChoiceQuestion(
    'Please select your favorite color (defaults to red)',
    ['red', 'blue', 'yellow'],
    0
);
$question->setErrorMessage('Color %s is invalid.');
```

```
$color = $helper->ask($input, $output, $question);
```


Commands: Pedir parámetros por pantalla.

Multiple Choice:

```
use Symfony\Component\Console\Question\ChoiceQuestion;
// ...
$helper = $this->getHelper('question');
$question = new ChoiceQuestion(
    'Please select your favorite colors (defaults to red and blue)',
    ['red', 'blue', 'yellow'],
    '0,1'
);
$question->setMultiselect(true);
$colors = $helper->ask($input, $output, $question);
```

Commands: Pedir parámetros por pantalla.

Autocompletar:

```
$helper = $this->getHelper('question');
```

```
$bundles = ['AcmeDemoBundle', 'AcmeBlogBundle', 'AcmeStoreBundle'];
```

```
$question = new Question('Please enter the name of a bundle', 'FooBundle');
```

```
$question->setAutocompleterValues($bundles);
```

```
$bundleName = $helper->ask($input, $output, $question);
```

Commands: Pedir parámetros por pantalla.

Ocultar la respuesta del usuario:

```
// ...
```

```
$helper = $this->getHelper('question');
```

```
$question = new Question('What is the database password?');
```

```
$question->setHidden(true);
```

```
$question->setHiddenFallback(false);
```

```
$password = $helper->ask($input, $output, $question);
```

Commands

RTFN:

<https://symfony.com/doc/current/console.html>

¿Preguntas?

