La consola de comandos

Para ver una lista de todos los comandos, basta con ejecutar simplemente bin/console o bin/console list

> *bin/console list*

El comando help seguido de un nombre de comando, nos proporciona la ayuda de dicho comando.

> *bin/console help list*

El último de los comandos genéricos de la consola es about, que nos da información sobre el proyecto actual

> *bin/console about*

Entorno de ejecución

Por defecto, los comandos de consola se ejecutan en el entorno definido en la variable de entorno APP_ENV.

Con los modificadores --env y --no-debug podemos modificar el comportamiento por defecto

```
bin/console comando
```

```
bin/console comando --env=prod

bin/console comando --env=test --no-debug
```

Comandos de consola personalizados

Para definir un comando, hay que crear una clase que extienda de **Command**.

Esta clase debe definir al menos dos métodos: **configure()** y **execute()**.

```
namespace App\Command;

use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputArgument;
use Symfony\Component\Console\Input\InputDefinition;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Input\InputOption;
use Symfony\Component\Console\Output\OutputInterface;

class DemoArgsCommand extends Command
{
    protected function configure()
    {
        // ...
    }

    protected function execute(InputInterface $input, Outp
utInterface $output)
```

```
    {
        // ...
    }
}
```

Configuración del comando

La configuración del comando se realiza dentro del método **execute()**.

```
namespace App\Command;

use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputArgument;
use Symfony\Component\Console\Input\InputDefinition;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Input\InputOption;
use Symfony\Component\Console\Output\OutputInterface;

class DemoArgsCommand extends Command
{
    protected function configure()
    {
        $this
            ->setName('demo:args')
            ->setDescription('Describe args behaviors')
            ->setDefinition(
                new InputDefinition(array(
```

```
                    new InputOption('foo', 'f'),
                    new InputOption('bar', 'b', InputOptio
n::VALUE_REQUIRED),
                    new InputOption('cat', 'c', InputOptio
n::VALUE_OPTIONAL),
                ))
            );
    }

    protected function execute(InputInterface $input, Outp
utInterface $output)
    {
        // ...
    }
}
```

La clase *Command* tiene disponibles varios métodos de configuración:

- setName()
- setDescription()
- setHelp()
- setDefinition()
- addArgument()
- addOption()

```
$this
        // El nombre del comando (la parte que va después
de "bin/console")
```

```
        ->setName('demo:args')

        // El texto mostrado cuando se ejecuta "bin/consol
e list"
        ->setDescription('Es un comando de prueba')

        // El texto mostrado cuando se ejecuta el comando
con la opción --help
        ->setHelp('Este comando te permite ...')
    ;
```

Registrar el comando

Una vez configurado, podemos registrarlo. Para registrarlo, se debe incluir como servicio con el tag **console.command**.

Si tenemos activado el *autoconfigure* se registrará automáticamente sin necesidad de incluirlo en el fichero services.yaml.

OPCIONES

Hay que distinguir entre opciones (InputOption) y argumentos (InputArgument), aunque luego comúnmente se mezclan los conceptos

Dentro del método **configure()**, las opciones se definen como se ve en el ejemplo anterior, dentro del **setDefinitio()** o también como se ve en el siguiente ejemplo, mediante el método **addOption()**:

```
$this
    // ...
    ->addOption(
        'iterations',
        'i',
        InputOption::VALUE_REQUIRED,
        'How many times should the message be printed?',
        1
    );
```

El primer parámetro es el nombre de la opción, el segundo es el nombre abreviado de la opción (puede ser null si no queremos forma abreviada), el tercero es el tipo de opción, el cuarto es el texto de ayuda sobre la opción y el quinto es el valor por defecto.

Únicamente el primer parámetro es obligatorio.

Al utilizar el comando, se especifican con **–opcion** o **-o** siendo *opcion* el nombre de la opcion y *o* el nombre abreviado de la opcion.

Las siguientes 6 formas de uso son equivalentes:

> *bin/console demo:args --iterations=5*

> *bin/console demo:args --iterations 5*

> *bin/console demo:args --iterations5*

> *bin/console demo:args -i=5*

> *bin/console demo:args -i 5*

> *bin/console demo:args -i5*

InputOption::VALUE_NONE

La opción *foo* puede existir o no existir, pero no se le puede pasar ningún valor. Si existe, su valor será *true*. Si no existe, su valor será *false*. El el tipo de opción por defecto si no se indica ningún tipo.

> *demo:args*

> *demo:args --foo*

> *demo:args -f*

InputOption::VALUE_REQUIRED

La opción *bar* necesita un valor obligatorio, por lo tanto, las tres ejecuciones anteriores darían error. El valor del argumento será el que le pasemos:

> *demo:args --bar=Hello*

> *demo:args --bar Hello*

> *demo:args -b=Hello*

> *demo:args -b Hello*

InputOption::VALUE_OPTIONAL

La opción *cat* puede existir o no existir. Pero en caso de existir se le debe pasar un valor.

Si existe y no se le pasa ningún valor, su valor acabará siendo *null*, lo mismo que si no hubiera existido.

InputOption::VALUE_IS_ARRAY

Esta opción acepta múltiples valores

> *demo:args --dir=/src --dir=/templates*

Se puede combinar VALUE_IS_ARRAY con VALUE_REQUIRED o con VALUE_OPTIONAL:

```
$this
    // ...
    ->addOption(
        'colors',
        null,
        InputOption::VALUE_REQUIRED | InputOption::VALUE_I
```

```
S_ARRAY,
        'Which colors do you like?',
        array('blue', 'red')
    );
```

ARGUMENTOS

Los argumentos son strings separados por espacios. Se asignar por orden de declaración.

Se pueden definir como las opciones o bien mediante el método **setDefinition()** o bien mediante el método **addArgument()**.

La definición admite hasta tres parámetros: el nombre del argumento, el tipo de argumento y el texto de ayuda sobre el argumento.

Únicamente el primer parámetro es obligatorio en la definición de un argumento.

```
// ...

new InputDefinition(array(
    // ...
    new InputArgument('arg1', InputArgument::REQUIRED),
    new InputArgument('arg2', InputArgument::IS_ARRAY),
));
```

InputArgument::OPTIONAL

El argumento es opcional. Es el comportamiento por defecto si no se indica el tipo de argumento.

InputArgument::REQUIRED

El comando no funciona si no se le pasa ningún valor a todos los argumentos obligatorios

InputArgument::IS_ARRAY

El argumento puede contener cualquier número de valores. Por esta razón, un argumento de este tipo debe de ser el último de la lista de argumentos.

Utilizando opciones y argumentos

Las siguientes dos ejecuciones del comando demo:args son equivalentes

> *demo:args World --bar Hello*

> *demo:args --bar Hello World*

En caso de necesidad, se puede utilizar el símbolo – para separar los argumentos de las opciones

> *demo:args --bar Hello --cat – World*

En este último ejemplo, la opción *cat* valdrá *null* y *World* será el valor del argumento.

Algoritmo del comando

DeSpués de configurar y registrar el comando, ya se puede ejecutar en la terminal:

> *bin/console demo:args --bar=Hello*

El código que se ejecuta al lanzar el comando es el código contenido en la función **execute()** de la clase.

Este método recibe instancias de objetos InputInterface y OutputInterface.

OutputInterface

La clase **OutputInterface** tiene dos métodos **write()** y **writeln()** para escribir en la salida. writeln() escribe el string pasado como argumento y termina con un salto de línea. Acepta ademas un array de strings.

```
// ...
protected function execute(InputInterface $input, OutputIn
terface $output)
```

```
{
    $output->writeln([
        'Comando demo',
        '============',
        '',
    ]);

    $output->writeln('¡Hola!');

    $output->write('Este es un comando ');
    $output->write('de prueba.');
}
```

InputInterface

La clase **InputInterface** tiene dos métodos getArgument() y getOption() para obtener los argumentos y las opciones pasadas al comando.

```
$output->writeln('Username: '.$input->getArgument('usernam
e'));
```

Inyección de dependencias

Al considerarse un servicio, cualquier comando, en su constructor, puede tipar argumentos para obtener instancias de otros servicios de la aplicación.

Ciclo de vida de un comando

Los comandos tienen 3 métodos que son invocados durante su ejecución:

- initialize()

Se ejecuta antes de interact() y de execute(). El propósito de este método es inicializar variables que necesiten el resto de métodos.

- interact()

Este método se ejecuta después de initialize() y antes de execute(). El propósito es comprobar si falta alguna de las opciones y/o argumentos y preguntarlas al usuario de forma interactiva. Después de este método si faltan argumentos u opciones obligatorias se producirá un error.

- execute()

Es el último en ejecutarse. Contiene la lógica del comando.

Console Helpers

El componente *console* dispone de una serie de “helpers” o pequeñas herramientas para ayudarnos con diferentes tareas muy comunes en los comandos.

- Question Helper: Para preguntar información al usuario de forma interactiva

https://symfony.com/doc/current/components/console/helpers/questionhelp

- Formatter Helper: Para personalizar colores de la salida

https://symfony.com/doc/current/components/console/helpers/formatterhel

- Progress Bar: Para mostrar una barra de progreso

https://symfony.com/doc/current/components/console/helpers/progressbar.

- Table: Para mostrar datos de forma tabulada

https://symfony.com/doc/current/components/console/helpers/table.html

El Helper Question

Vamos a empezar con un ejemplo en el que realizamos al usuario una pregunta de sí o no:

```
// ...
```

```
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use Symfony\Component\Console\Question\ConfirmationQuestio
n;

class YourCommand extends Command
{
    // ...

    public function execute(InputInterface $input, OutputI
nterface $output)
    {
        $helper = $this->getHelper('question');
        $question = new ConfirmationQuestion('¿Estás segur
o de que quieres realizar esta operación?', false, '/^(y|s
)/i');

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

La forma interactuar con el usuario es a través de objetos **Question**. En este caso, crearemos una **ConfirmationQuestion** que permite al usuario responder *yes* o *no*.

El segundo parámetro del constructor de ConfirmationQuestion es el valor por defecto si el usuario no provee ninguna respuesta.

El tercer parámetro permite personalizar el patrón utilizado para considerar que la respuesta es afirmativa. Por ejemplo en este caso, cualquier cosa que empiece con y o con s se consideraría afirmativo.

Obtenemos una instancia del helper y utilizamos su método **ask()** para presentar la Question al usuario. El método ask() devuelve la respuesta del usuario.

Otros objetos Question

- Question

```
use Symfony\Component\Console\Question\Question;

// ...
public function execute(InputInterface $input, OutputInter
face $output)
{
    // ...
    $question = new Question('Introduce el nombre de la en
tidad', 'User');

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

- ChoiceQuestion

```
use Symfony\Component\Console\Question\ChoiceQuestion;

// ...
public function execute(InputInterface $input, OutputInter
face $output)
{
    // ...
    $helper = $this->getHelper('question');
    $question = new ChoiceQuestion(
        'Please select your favorite color (defaults to re
d)',
        array('red', 'blue', 'yellow'),
        0
    );
    $question->setErrorMessage('Color %s is invalid.');

    $color = $helper->ask($input, $output, $question);
    $output->writeln('You have just selected: '.$color);

    // ... do something with the color
}
```

Con variante multiselección:

```
use Symfony\Component\Console\Question\ChoiceQuestion;
```

```
// ...
public function execute(InputInterface $input, OutputInter
face $output)
{
    // ...
    $helper = $this->getHelper('question');
    $question = new ChoiceQuestion(
        'Please select your favorite colors (defaults to r
ed and blue)',
        array('red', 'blue', 'yellow'),
        '0,1'
    );
    $question->setMultiselect(true);

    $colors = $helper->ask($input, $output, $question);
    $output->writeln('You have just selected: ' . implode(
', ', $colors));
}
```

Autocompletado

```
use Symfony\Component\Console\Question\Question;

// ...
public function execute(InputInterface $input, OutputInter
face $output)
```

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

    $bundles = array('AcmeDemoBundle', 'AcmeBlogBundle', '
AcmeStoreBundle');
    $question = new Question('Please enter the name of a b
undle', 'FooBundle');
    $question->setAutocompleterValues($bundles);

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

Esconder la respuesta del usuario

```
use Symfony\Component\Console\Question\Question;

// ...
public function execute(InputInterface $input, OutputInter
face $output)
{
    // ...
    $helper = $this->getHelper('question');

    $question = new Question('Introduzca la contraseña: ')
;
```

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

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

Normalizar la respuesta del usuario

```
use Symfony\Component\Console\Question\Question;

// ...
public function execute(InputInterface $input, OutputInter
face $output)
{
    // ...
    $helper = $this->getHelper('question');

    $question = new Question('Please enter the name of the
 bundle', 'AcmeDemoBundle');
    $question->setNormalizer(function ($value) {
        // $value can be null here
        return $value ? trim($value) : '';
    });

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

```
}
```

Validar la respuesta del usuario

```
use Symfony\Component\Console\Question\Question;

// ...
public function execute(InputInterface $input, OutputInter
face $output)
{
    // ...
    $helper = $this->getHelper('question');

    $question = new Question('Please enter the name of the
 bundle', 'AcmeDemoBundle');
    $question->setValidator(function ($answer) {
        if (!is_string($answer) || 'Bundle' !== substr($an
swer, -6)) {
            throw new \RuntimeException(
                'The name of the bundle should be suffixed
 with \'Bundle\''
            );
        }

        return $answer;
    });
    $question->setMaxAttempts(2);
```

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