Console

The way you can create useful tasks, crons, and batch jobs

By https://andreiabohner.org

Symfony 4.2

Installation

```
$ composer require symfony/console
```

Using the console

[arguments] and [options] allow to pass dynamic information from the terminal to the command

The strings - separated by spaces - that come after the command name itself. They are ordered, and can be optional or required

Are not ordered (you can specify them in any order) and are specified with two dashes (e.g.: --env)

Usage (syntax)

```
$ php bin/console <command> [arguments] [options]
```

E.g.: `$ php bin/console debug:container 'App\Service\Mailer' --show-arguments`

Common to all commands

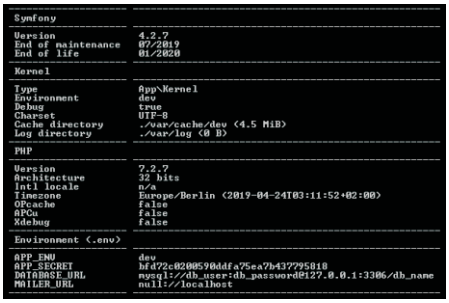
Options

`-h, --help`	Display this help message
`-q, --quiet`	Do not output any message
`-V, --version`	Display this application version
`--ansi`	Force ANSI output
`--no-ansi`	Disable ANSI output
`-n, --no-interaction`	Do not ask any interactive question
`-e, --env=ENV`	The Environment name. [default: "dev"]
`--no-debug`	Switches off debug mode.
`-v|vv|vvv, --verbose`	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Console commands run in the environment defined in the APP_ENV variable of the **.env** file, which is **dev** by default. It also reads the APP_DEBUG value to turn "**debug**" mode on or off (it defaults to 1, which is **on**).

General commands

`about` — Displays information about the current project



`help` — Displays help for a specific command

```
$ php bin/console help doctrine:database:create
```

`list` — View the command list

```
$ php bin/console list doctrine
```

List all available commands

```
$ php bin/console
```

Some common commands available

(depends on the libraries you have installed)

Assets

`assets:install`	Installs bundles web assets under a public directory
`<target-directory>`	
`--symlink`	Symlinks the assets instead of copying it
`--relative`	Make relative symlinks
`--no-cleanup`	Do not remove the assets of the bundles that no longer exist

Cache

`cache:clear`	Clears the cache
`--no-warmup`	Do not warm up the cache
`--no-optional-warmers`	Skip optional cache warmers (faster)
`cache:pool:clear`	Clears cache pools
`<list-of-cache-pools>`	
`cache:pool:delete`	Deletes an item from a cache pool
`<cache-pool> <cache-key-to-delete>`	
`cache:pool:prune`	Prunes cache pools
`cache:warmup`	Warms up an empty cache
`--no-optional-warmers`	Skip optional cache warmers (faster)

Config

`config:dump-reference`	Dumps the default configuration for an extension
`<BundleNameOrExtensionAlias> <configuration-option-path>`	
`--format=FORMAT`	The output format (yaml or xml) [default: "yaml"]

Console

Symfony 4.2

Doctrine

```
doctrine:cache:clear-collection-region                Clear a second-level cache collection region
    <OwnerEntity> <collectionAssociationName> <owner-id>
    --all                                             All entity regions will be deleted/invalidated
    --flush                                           All cache entries will be flushed
    --em[=EM]                                         The entity manager to use for this command
doctrine:cache:clear-entity-region                    Clear a second-level cache entity region
    <EntityClass> <entity-id>
    --all                                             All entity regions will be deleted/invalidated
    --flush                                           All cache entries will be flushed
    --em[=EM]                                         The entity manager to use for this command
    E.g.: doctrine:cache:clear-entity-region 'Entities\MyEntity' 1
doctrine:cache:clear-metadata                         Clears all metadata cache for an entity manager
    --flush                                           Cache entries will be flushed instead of deleted/invalidated.
    --em[=EM]                                         The entity manager to use for this command
doctrine:cache:clear-query                            Clears all query cache for an entity manager
    --flush                                           Cache entries will be flushed instead of deleted/invalidated.
    --em[=EM]                                         The entity manager to use for this command
doctrine:cache:clear-query-region                     Clear a second-level cache query region
    <region-name>
    --all                                             All entity regions will be deleted/invalidated
    --flush                                           All cache entries will be flushed
    --em[=EM]                                         The entity manager to use for this command
doctrine:cache:clear-result                           Clears result cache for an entity manager
    --flush                                           Cache entries will be flushed instead of deleted/invalidated.
    --em[=EM]                                         The entity manager to use for this command
doctrine:cache:contains                               Check if a cache entry exists
    <cache-name> <cache-id>
doctrine:cache:delete                                 Delete a cache entry
    <cache-name> <cache-id>
doctrine:cache:flush                                  Flush a given cache
or: doctrine:cache:clear
    <cache-name>
doctrine:cache:stats                                  Get stats on a given cache provider
    <cache-name>
doctrine:database:create                              Create database using the db_name of your  database connection information
    --shard=SHARD                                     The shard connection to use for this command
    --connection[=CONNECTION]                         The connection to use for this command
    --if-not-exists                                   Don't trigger an error, when the database already exists
doctrine:database:drop                                Drops the configured database
    --shard=SHARD                                     The shard connection to use for this command
    --connection[=CONNECTION]                         The connection to use for this command
    --if-exists                                       Don't trigger an error, when the database doesn't exists
doctrine:database:import                              Import SQL file(s) directly to Database.
    <file-path-sql-to-execute>
    --connection[=CONNECTION]                         The connection to use for this command
doctrine:ensure-production-settings                   Verify that Doctrine is properly configured for a production environment
    --complete                                        Flag to also inspect database connection existence.
    --em[=EM]                                         The entity manager to use for this command
doctrine:generate:entities                            Generates entity classes and method stubs from your mapping information
or: generate:doctrine:entities
    <BundleName-or-EntityName-or-NamespaceName>
    --path=PATH                                       The path where to generate entities when it cannot be guessed
    --no-backup                                       Do not backup existing entities files.
    E.g.: php bin/console doctrine:generate:entities Blog/Entity --path=src/
```

Console

Symfony 4.2

Doctrine

`doctrine:mapping:convert`	Convert mapping information between supported formats
`or: orm:convert:mapping`	
`<type-to-be-converted> <path-to-generate-entities-classes>`	
`--filter=FILTER`	A string pattern used to match entities that should be processed. (multiple values allowed)
`-f, --force`	Force to overwrite existing mapping files
`--from-database`	Whether or not to convert mapping information from existing database.
`--extend[=EXTEND]`	Defines a base class to be extended by generated entity classes
`--num-spaces[=NUM-SPACES]`	Defines the number of indentation spaces [default: 4]
`--namespace[=NAMESPACE]`	Defines a namespace for the generated entity classes, if converted from database
`--em[=EM]`	The entity manager to use for this command
`doctrine:mapping:import`	Imports mapping information from an existing database
`<BundleOrNamespaceToImport> <mapping-type-to-export>`	
`--em[=EM]`	The entity manager to use for this command
`--shard=SHARD`	The shard connection to use for this command
`--filter=FILTER`	A string pattern used to match entities that should be mapped. (multiple values allowed)
`--force`	Force to overwrite existing mapping files.
`--path=PATH`	The path where the files would be generated (not used when a bundle is passed).
`E.g.: php bin/console doctrine:mapping:import "MyBundle" annotation`	
`doctrine:mapping:info`	Show entities that Doctrine is aware of and whether or not there are any basic errors with the mapping
`--em[=EM]`	The entity manager to use for this command
`doctrine:migrations:diff`	Generate a migration by comparing your current database to your mapping information
`or: diff`	
`--configuration[=CONFIGURATION]`	The path to a migrations configuration file
`--db-configuration[=DB-CONFIGURATION]`	The path to a database connection configuration file
`--editor-cmd[=EDITOR-CMD]`	Open file with this command upon creation
`--filter-expression[=FILTER-EXPRESSION]`	Tables which are filtered by Regular Expression
`--formatted`	Format the generated SQL
`--line-length[=LINE-LENGTH]`	Max line length of unformatted lines. [default: 120]
`--db=DB`	The database connection to use for this command
`--em[=EM]`	The entity manager to use for this command
`--shard=SHARD`	The shard connection to use for this command
`doctrine:migrations:dump-schema`	Dump the schema for your database to a migration.
`or: dump-schema`	
`--configuration[=CONFIGURATION]`	The path to a migrations configuration file
`--db-configuration[=DB-CONFIGURATION]`	The path to a database connection configuration file
`--editor-cmd[=EDITOR-CMD]`	Open file with this command upon creation
`--formatted`	Format the generated SQL
`--line-length[=LINE-LENGTH]`	Max line length of unformatted lines. [default: 120]
`--db=DB`	The database connection to use for this command
`--em[=EM]`	The entity manager to use for this command
`--shard=SHARD`	The shard connection to use for this command
`doctrine:migrations:execute`	Execute a single migration version up or down manually
`or: execute`	
`<version-to-execute>`	
`--write-sql[=WRITE-SQL]`	The path to output the migration SQL file instead of executing it. Defaults to current working directory. [default: false]
`--dry-run`	Execute the migration as a dry run
`--up`	Execute the migration up
`--down`	Execute the migration down
`--query-time`	Time all the queries individually
`--configuration[=CONFIGURATION]`	The path to a migrations configuration file
`--db-configuration[=DB-CONFIGURATION]`	The path to a database connection configuration file
`--db=DB`	The database connection to use for this command
`--em=EM`	The entity manager to use for this command
`--shard=SHARD`	The shard connection to use for this command
`E.g.: php bin/console doctrine:migrations:execute YYYYMMDDHHMMSS --down`	

Console

Symfony 4.2

Doctrine

`doctrine:migrations:generate`
Generate a blank migration class.

`or: generate`

`--editor-cmd[=EDITOR-CMD]`	Open file with this command upon creation
`--configuration[=CONFIGURATION]`	The path to a migrations configuration file
`--db-configuration[=DB-CONFIGURATION]`	The path to a database connection configuration file
`--db=DB`	The database connection to use for this command
`--em=EM`	The entity manager to use for this command
`--shard=SHARD`	The shard connection to use for this command

`doctrine:migrations:latest`
Outputs the latest version number

`or: latest`

`--configuration[=CONFIGURATION]`	The path to a migrations configuration file
`--db-configuration[=DB-CONFIGURATION]`	The path to a database connection configuration file
`--db=DB`	The database connection to use for this command
`--em=EM`	The entity manager to use for this command
`--shard=SHARD`	The shard connection to use for this command

`doctrine:migrations:migrate`
Execute a migration to a specified version or the latest available version.

`or: migrate`

`<version-number-or-alias-to-migrate(first, prev, next, latest)>` [default: latest]

`--write-sql[=WRITE-SQL]`	The path to output the migration SQL file instead of executing it. Defaults to current working directory.
`--dry-run`	Execute the migration as a dry run
`--query-time`	Time all the queries individually [default: false]
`--allow-no-migration`	Don't throw an exception if no migration is available (CI)
`--all-or-nothing[=ALL-OR-NOTHING]`	Wrap the entire migration in a transaction. [default: false]
`--configuration[=CONFIGURATION]`	The path to a migrations configuration file
`--db-configuration[=DB-CONFIGURATION]`	The path to a database connection configuration file
`--db=DB`	The database connection to use for this command
`--em=EM`	The entity manager to use for this command
`--shard=SHARD`	The shard connection to use for this command

E.g.: `php bin/console doctrine:migrations:migrate current+3`

`doctrine:migrations:rollup`
Rollup migrations by deleting all tracked versions and insert the one version that exists

`or: rollup`

`--configuration[=CONFIGURATION]`	The path to a migrations configuration file
`--db-configuration[=DB-CONFIGURATION]`	The path to a database connection configuration file
`--db=DB`	The database connection to use for this command
`--em=EM`	The entity manager to use for this command
`--shard=SHARD`	The shard connection to use for this command

`doctrine:migrations:status`
View the status of a set of migrations

`or: status`

`--show-versions`	This will display a list of all available migrations and their status
`--configuration[=CONFIGURATION]`	The path to a migrations configuration file
`--db-configuration[=DB-CONFIGURATION]`	The path to a database connection configuration file
`--db=DB`	The database connection to use for this command
`--em=EM`	The entity manager to use for this command
`--shard=SHARD`	The shard connection to use for this command

`doctrine:migrations:up-to-date`
Tells you if your schemais up-to-date

`or: up-to-date`

`--configuration[=CONFIGURATION]`	The path to a migrations configuration file
`--db-configuration[=DB-CONFIGURATION]`	The path to a database connection configuration file
`--db=DB`	The database connection to use for this command
`--em=EM`	The entity manager to use for this command
`--shard=SHARD`	The shard connection to use for this command

Console

Symfony 4.2

Doctrine

`doctrine:migrations:version` — Manually add and delete migration versions from the version table

or: version

- `<version-to-add-or-delete>`
- `--add` — Add the specified version
- `--delete` — Delete the specified version
- `--all` — Apply to all the versions
- `--range-from[=RANGE-FROM]` — Apply from specified version
- `--range-to[=RANGE-TO]` — Apply to specified version
- `--configuration[=CONFIGURATION]` — The path to a migrations configuration file
- `--db-configuration[=DB-CONFIGURATION]` — The path to a database connection configuration file
- `--db=DB` — The database connection to use for this command
- `--em=EM` — The entity manager to use for this command
- `--shard=SHARD` — The shard connection to use for this command

E.g.1: `php bin/console doctrine:migrations:version YYYYMMDDHHMMSS --add`

E.g.2: `php bin/console doctrine:migrations:version --add --all` Skip all existing migrations if you want to create your schema via doctrine:schema:create

`doctrine:query:dql` — Executes arbitrary DQL directly from the command line

- `<DQL-to-execute>`
- `--hydrate=HYDRATE` — Hydration mode of result set. Should be either: object, array, scalar or single-scalar. [default: "object"]
- `--first-result=FIRST-RESULT` — The first result in the result set
- `--max-result=MAX-RESULT` — The maximum number of results in the result set
- `--depth=DEPTH` — Dumping depth of Entity graph. [default: 7]
- `--show-sql` — Dump generated SQL instead of executing query
- `--em[=EM]` — The entity manager to use for this command

`doctrine:query:sql` — Executes arbitrary SQL directly from the command line.

- `<SQL-to-execute>`
- `--depth=DEPTH` — Dumping depth of result set. [default: 7]
- `--force-fetch` — Forces fetching the result.
- `--connection[=CONNECTION]` — The connection to use for this command

E.g.: `php bin/console doctrine:query:sql "SELECT * FROM users"`

`doctrine:schema:create` — Executes (or dumps) the SQL needed to generate the database schema

- `--dump-sql` — Instead of trying to apply generated SQLs into EntityManager Storage Connection, output them
- `--em[=EM]` — The entity manager to use for this command

`doctrine:schema:drop` — Executes (or dumps) the SQL needed to drop the current database schema

- `--dump-sql` — Instead of trying to apply generated SQLs into EntityManager Storage Connection, output them
- `-f, --force` — Don't ask for the deletion of the database, but force the operation to run
- `--full-database` — Instead of using the Class Metadata to detect the DB table schema, drop ALL assets that the DB contains
- `--em[=EM]` — The entity manager to use for this command

`doctrine:schema:update` — Executes (or dumps) the SQL needed to update the DB schema to match the current mapping metadata

- `--complete` — All assets of the database which are not relevant to the current metadata will be dropped
- `--dump-sql` — Dumps the generated SQL statements to the screen (does not execute them)
- `-f, --force` — Causes the generated SQL statements to be physically executed against your database
- `--em[=EM]` — The entity manager to use for this command

`doctrine:schema:validate` — Validate the mapping files

- `--skip-mapping` — Skip the mapping validation check
- `--skip-sync` — Skip checking if the mapping is in sync with the database
- `--em[=EM]` — The entity manager to use for this command

Console

Symfony 4.2

Swiftmailer

`swiftmailer:email:send`	Send simple email message
`--from=FROM`	The from address of the message
`--to=TO`	The to address of the message
`--subject=SUBJECT`	The subject of the message
`--body=BODY`	The body of the message
`--mailer=MAILER`	The mailer name [default: "default"]
`--content-type=CONTENT-TYPE`	The body content type of the message [default: "text/html"]
`--charset=CHARSET`	The body charset of the message [default: "UTF8"]
`--body-source=BODY-SOURCE`	The source where body come from [stdin|file] [default: "stdin"]

```
E.g.: php bin/console swiftmailer:email:send --body-source=file --body=/path/to/file
```

`swiftmailer:spool:send`	Sends emails from the spool
`--message-limit=MESSAGE-LIMIT`	The maximum number of messages to send
`--time-limit=TIME-LIMIT`	The time limit for sending messages (in seconds)
`--recover-timeout=RECOVER-TIMEOUT`	The timeout for recovering messages that have taken too long to send (in seconds)
`--mailer=MAILER`	The mailer name
`--transport=TRANSPORT`	The service of the transport to use tosend the messages

```
E.g.: php bin/console swiftmailer:spool:send --message-limit=10 --time-limit=10 --recover-timeout=900 --mailer=default
```

Translation

`translation:update`	Updates the translation file
`<locale> <BundleNameToLoadMessages>`	
`--prefix[=PREFIX]`	Override the default prefix [default: "__"]
`--output-format[=OUTPUT-FORMAT]`	Override the default output format [default: "xlf"]
`--dump-messages`	Should the messages be dumped in the console
`--force`	Should the update be done
`--no-backup`	Should backup be disabled
`--clean`	Should clean not found messages
`--domain[=DOMAIN]`	Specify the domain to update

```
E.g.: php bin/console translation:update --force --prefix="new_" fr
      php bin/console translation:update --dump-messages en AcmeBundle
```

Lint

`lint:twig`	Lints a template and outputs encountered errors
`<filename>`	
`--format=FORMAT`	The output format [default: "txt"]

```
E.g: php bin/console lint:twig @AcmeDemoBundle                  Lint all template files in a bundle
     cat filename | php bin/console lint:twig                   Validate the syntax of contents passed from STDIN
     php bin/console lint:twig dirname --format=json           Validate a whole directory
```

`lint:xliff`	Lints a XLIFF file and outputs encountered errors
`<filename>`	
`--format=FORMAT`	The output format [default: "txt"]

```
E.g: php bin/console lint:xliff @AcmeDemoBundle                 Lint all template files in a bundle
     cat filename | php bin/console lint:xliff                  Validate the syntax of contents passed from STDIN
     php bin/console lint:xliff dirname --format=json          Validate a whole directory
```

`lint:yaml`	Lints a file and outputs encountered errors
`<filename>`	
`--format=FORMAT`	The output format [default: "txt"]

```
E.g: php bin/console lint:yaml @AcmeDemoBundle                  Lint all template files in a bundle
     cat filename | php bin/console lint:yaml                   Validate the syntax of contents passed from STDIN
     php bin/console lint:yaml dirname --format=json           Validate a whole directory
```

Console

Symfony 4.2

Router

`router:match`	Helps debug routes by simulating a path info match
`<path-info>`	
`--method=METHOD`	Sets the HTTP method
`--scheme=SCHEME`	Sets the URI scheme (usually http or https)
`--host=HOST`	Sets the URI host

E.g.: `php bin/console router:match /foo --method POST --scheme https --host symfony.com --verbose`

Make

`make:auth`	Creates a Guard authenticator of diferent flavors
`make:command`	Creates a new console command class
`<name-of-command>`	
`make:controller`	Creates a new controller class
`<name-of-controller-class>`	
`make:crud`	Creates CRUD for Doctrine entity class
`<name-of-entity-class>`	
`make:entity`	Creates or updates a Doctrine entity class, and optionally an API Platform resource
`<name-of-entity-to-create>`	
`-a, --api-resource`	Mark this class as an API Platform resource (expose a CRUD API for it)
`--regenerate`	Instead of adding new fields, simply generate the methods (e.g. getter/setter) for existing fields
`--overwrite`	Overwrite any existing getter/setter methods
`make:fixtures`	Creates a new class to load Doctrine fixtures
`<name-of-fixture-class>`	
`make:form`	Creates a new form class
`<name-of-form-class> <name-of-entity-bound-to>`	
`make:functional-test`	Creates a new functional test class
`<name-of-functional-test-class>`	
`make:migration`	The database connection to use for this command
`--db=DB`	The entity manager to use for this command
`--em=EM`	The shard connection to use for this command
`--shard=SHARD`	Creates a new migration based on database changes
`make:registration-form`	Creates a new registration form system
`make:serializer:encoder`	Creates a new serializer encoder class
`<name> <format>`	

E.g. `php bin/console make:serializer:encoder YamlEncoder yaml`

`make:serializer:normalizer`	Creates a new serializer normalizer class
`<name>`	
`make:subscriber`	Creates a new event subscriber class
`<name> <event>`	
`make:twig-extension`	Creates a new Twig extension class
`<name>`	
`make:unit-test`	Creates a new unit test class
`<name>`	
`make:user`	Creates a new security user class
`<name>`	
`--is-entity`	Do you want to store user data in the database (via Doctrine)?
`--identity-property-name=IDENTITY-PROPERTY-NAME`	Enter a property name that will be the unique "display" name for the user (e.g. email, username)
`--with-password`	Will this app be responsible for checking the password? Choose No if the password is actually checked by some other system (e.g. a single sign-on server)
`--use-argon2`	Use the Argon2i password encoder?
`make:validator`	Creates a new validator and constraint class
`<name>`	
`make:voter`	Creates a new security voter class
`<name>`	

Console

Symfony 4.2

Debug

debug:autowiring — Show all classes/interfaces you can use for autowiring

```
<search-filter>
--all                    Show also services that are not aliased
```

debug:config — Dumps the current configuration for an extension

```
<bundle-name-or-extension-alias> <configuration-option-path>
E.g.: php bin/console debug:config framework serializer.enabled
```

debug:container — Displays full list of current services available in the container

```
<service-name>
--show-private           Used to show public *and* private services (deprecated)
--show-arguments         Used to show arguments in services
--show-hidden            Used to show hidden (internal) services (whose ID starts with a dot)
--tag=TAG                Shows all services with a specific tag
--tags                   Displays tagged services for an application
--parameter=PARAMETER    Displays a specific parameter for an application
--parameters             Displays parameters for an application
 --types                 Displays types (classes/interfaces) available in the container
--format=FORMAT          The output format (txt, xml, json, or md) [default: "txt"]
--raw                    To output raw description
E.g.: php debug:container 'App\Service\Mailer'   Detailed info about a single service (you can use the service id too)
```

debug:event-dispatcher — Displays configured listeners for an application

```
<event-name>
--format=FORMAT          The output format (txt, xml, json, or md) [default: "txt"]
--raw                    To output raw description
```

debug:form — Displays form type information

```
<form-type-class> <form-type-option>
--show-deprecated        Display deprecated options in form types
--format=FORMAT          The output format (txt or json) [default: "txt"]
```

debug:router — Displays current routes for an application

```
<route-name>
--show-controllers       Show assigned controllers in overview
--format=FORMAT          The output format (txt, xml, json, or md) [default: "txt"]
--raw                    To output raw route(s)
```

debug:swiftmailer — Displays current mailers for an application

```
<mailer-name>
```

debug:translation — Displays translation messages information

```
<locale> <BundleNameToLoadMessages>
--domain[=DOMAIN]        The messages domain
--only-missing           Displays only missing messages
--only-unused            Displays only unused messages
--all                    Load messages from all registered bundles
E.g.: php bin/console debug:translation --only-unused en AcmeDemoBundle
```

debug:twig] — Shows a list of twig functions, filters, globals and tests

```
<template-name>
--filter=FILTER          Show details for all entries matching this filter
--format=FORMAT          The output format (text or json) [default: "text"]
E.g.: php bin/console debug:twig @Twig/Exception/error.html.twig   Lists all paths that match the given template name
      php bin/console debug:twig --filter=date                     Lists everything that contains the word date
```

Console

Symfony 4.2

Security

`security:encode-password` — Encodes a password

`<password-to-encode> <UserEntityClassPathAssociatedWithEncoder>`

`--empty-salt` — Do not generate a salt or let the encoder generate one

E.g.: `php bin/console security:encode-password --no-interaction [password] App\Entity\User`

Server

`server:dump` — Starts a dump server that collects and displays dumps in a single place

`--format=FORMAT` — The output format (cli, html) [default: "cli"]

`server:log` — Starts a log server that displays logs in real time

`--host=HOST` — The server host [default: "0.0.0.0:9911"]

`--format=FORMAT` — The line format [default: "%datetime% %start_tag%%level_name%%end_tag% <comment>[%channel%]</> %message%%context%%extra%\n"]

`--date-format=DATE-FORMAT` — The date format [default: "H:i:s"]

`--filter=FILTER` — An expression to filter log. Example: "level >200 or channel in ['app', 'doctrine']"

E.g.: `php bin/console server:log --filter=port`

`server:run` — Runs a local web server

`<address-port-to-listen>`

`-d, --docroot=DOCROOT` — Document root, usually where your front controllers are stored

`-r, --router=ROUTER` — Path to custom router script

E.g.: `php bin/console server:run 127.0.0.1:8080`

`server:start` — Starts a local web server in the background

`<address-port-to-listen>`

`-d, --docroot=DOCROOT` — Document root, usually where your front controllers are stored

`-r, --router=ROUTER` — Path to custom router script

`--pidfile=PIDFILE` — PID file

E.g.: `php bin/console server:start --docroot=htdocs/`

`php bin/console server:start --router=app/config/router.php`

`server:status` — Outputs the status of the local web server

`--pidfile=PIDFILE` — PID file

`--filter=FILTER` — The value to display (one of port, host, or address)

E.g.: `php bin/console server:status --filter=port`

`server:stop` — Stops the local web server that was started with the server:start command

`--pidfile=PIDFILE` — PID file

Console

Symfony 4.2

Creating a Command

```php
// src/Command/CreateUserCommand.php
namespace App\Command;

use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use App\Service\UserManager;

class CreateUserCommand extends Command
{

    protected static $defaultName = 'app:create-user';
    private $userManager;

    public function __construct(UserManager $userManager)
    {
        $this->userManager = $userManager;
        parent::__construct();
    }

    protected function configure()
    {
        $this
            ->setDescription('Creates a new user.')
            ->setHelp('This command allows you to create a user...')
            ->addArgument('username', InputArgument::REQUIRED, 'What is the username?')
            ->addArgument('last_name', InputArgument::OPTIONAL, 'User last name?')
        ;
    }

    protected function execute(InputInterface $input, OutputInterface $output)
    {
        // the logic of the command
        // ...

       $this->userManager->create($input->getArgument('username'));

        // retrieve the argument value using getArgument()
        $output->writeln('Username: '.$input->getArgument('username'));
    }
}
```

Commands are defined in classes extending Command

name of the command (the part after "bin/console")

use normal autowiring to get services

Don't forget to call the parent constructor

is called automatically at the end of the command constructor

Optional — *the full command description shown when running the command with the "--help" option*

Optional — *the short description shown while running "php bin/console list"*

Arguments are the strings - separated by spaces - that come after the command name itself. They are ordered, and can be optional or required.

outputs content to the console

Methods that are Invoked When Running a Command

`initialize()`	*Optional*	Executed before the interact() and the execute() methods. Purpose: initialize variables used in the rest of the command methods.
`interact()`	*Optional*	Executed after initialize() and before execute(). Purpose: check if some of the options/arguments are missing and interactively ask the user for those values. This is the last place where you can ask for missing options/arguments. After this command, missing options/arguments will result in an error.
`execute()`	*Required*	Executed after interact() and initialize(). It contains the logic you want the command to execute.