Table of Contents

Drupal Console Documentation

> **Note: This project is a work-in-progress.**

This book documents the Drupal Console project.

Contribute to the project

You can contribute to improve this project on Github

Contribute to this documentation

You can contribute to improve this documentation on GitHub.

> Drupal is a registered trademark of Dries Buytaert.

What is the Drupal Console?

The Drupal CLI. A tool to generate boilerplate code, interact with and debug Drupal. From the ground up, it has been built to utilize the same modern PHP practices which were introduced in Drupal 8.

Drupal Console started as an scaffolding tool but evolved into a full CLI with the help of multiple contributors.

The Drupal Console makes use of the Symfony Console and other third party components which allows you to automatically generate most of the code needed for a Drupal 8 module. In addition, Drupal Console helps you interact with your Drupal installation.

Why should you care about it

Drupal 8 is more technically advanced compared to its predecessor and managing the increasing complexity of Drupal 8 could be a daunting task for anyone. Drupal Console is a suite of tools to help manage that complexity. Writing a Drupal 8 module now involves a lot more boilerplate code and there is a lot you need to know and do just to *get started* building a new module. These tasks can be repetitive and tedious, and can therefore create increased potential for errors. Fortunately, a lot of the new code can be automatically generated, using Drupal Console, without risk of copy/paste errors and with a lot of saved time.

Benefits of the Drupal Console:

- Takes advantage of the Symfony Console and other third-party components to generate PHP, YML, and other files.
- Takes advantage of other modern development practices.
- Saves development time, both during migration of existing Drupal modules and when writing new ones.
- Provides easy-to-learn tools that make Drupal 8 development, by extension, also easier to learn.
- Reduces development time for remaining Drupal 8 tasks and for development of new modules.

Follow along in this documentation as we explore the power of this exciting new set of tools.

How does Drupal Console help?

Generating the code and files required by a Drupal 8 module

Drupal Console provides a number of commands for creating module scaffolding and boilerplate code. For any command, you will be asked a series of questions about what you want to generate. Based on that user interaction, it will then generate the required boilerplate to build the requested component.

Introspecting the system.

Drupal Console allows you to debug routes, services, plugins, configurations, events and other components and subsystems.

Interacting with your Drupal installation.

Drupal Console help you to interact with your Drupal installation, from rebuilding cache, importing/exporting configuration, and resetting passwords among others.

Learning Drupal 8

Drupal Console helps you learn Drupal 8. In addition to generating complex code, you can increase the verbosity of the code comments, to better understand the generated code and how to build on it, by using the `--learning` option.

Where do I find the project?

Project landing page

http://drupalconsole.com

Github repository

https://github.com/hechoendrupal/drupal-console

Documentation

https://docs.drupalconsole.com/

Support chat

https://gitter.im/hechoendrupal/DrupalConsole

More information on Drupal.org project page

https://drupal.org/project/console

Getting the project

You need to install two things to get DrupalConsole working:

1. The DrupalConsole Launcher
2. DrupalConsole itself

Why do I need the Launcher?

This is a global executable that enables you to run the command, `drupal`, from any directory within your site's project. Without it you will be inconvenienced by having to run the command only from your drupal root directory.

For example, if you have Drupal root in a /web directory, and a composer.json and your vendor directory in the directory above that, you will be able to run the `drupal` command from the same directory as the composer.json file. Even better, you can run it from any subdirectory under that as many levels deep as you would like to go.

Install Drupal Console Launcher aka Global executable

Install DrupalConsole in each one of your projects using Composer

Each one of your site projects should have it's own DrupalConsole installed. This is done using Composer.

Install Drupal Console Using Composer

Notes: Starting on RC releases DrupalConsole must be installed per site. Install Drupal Console using `composer global require` is no longer supported.

Help!

See the FAQ section below (section 7) for help with specific installation and command issues.

Install Drupal Console Using Composer

Change directory to Drupal site:

```
cd /path/to/drupal8.dev
```

Execute composer require command:

```
composer require drupal/console:~1.0 \
--prefer-dist \
--optimize-autoloader
```

Download using DrupalComposer project template

```
composer create-project \
drupal-composer/drupal-project:8.x-dev \
drupal8.dev \
--prefer-dist \
--no-progress \
--no-interaction
```

Update DrupalConsole

```
composer update drupal/console --with-dependencies
```

Install Drupal Console Launcher using the phar

```
curl https://drupalconsole.com/installer -L -o drupal.phar
mv drupal.phar /usr/local/bin/drupal
chmod +x /usr/local/bin/drupal
```

NOTE: If you don't have curl you can try

```
php -r "readfile('https://drupalconsole.com/installer');" > drupal.phar
```

Update DrupalConsole Launcher

```
drupal self-update
```

Run Drupal Console using the Launcher

```
drupal
```

Installing the Launcher using Composer

Set Composer global `minimum-stability` and `prefer-stable` configurations.

```
composer config minimum-stability dev
composer config prefer-stable true
```

```
composer global require drupal/console-launcher:~1.0
```

You must execute the launcher within a drupal site directory or use `--root=/path/to/drupal8.dev` to specify your drupal site path.

NOTE: The name `drupal` is just an alias you can name it anything you like.

Installing Drupal Console on Windows

Here is a link to an excellent article about Installing Drupal Console on Windows using Acquia Dev Desktop.

When using Windows you will need some extra tools.

CLI tools

- Console emulator for Windows
- Git for Windows. Provides a BASH emulator.
- Composer

Local server Stack

- Acquia Dev Desktop
- XAMP
- WAMP
- MAMP

Virtual Environments

- DrupalVM
- Docksal
- Lando
- Kalabox

Using the project

Drupal Console provides two types of commands.

1. **Global Launcher Commands:** These commands can run outside of a Drupal 8 site root.
2. **Per-site Commands:** These commands must be run within a Drupal 8 site root.

Executing Drupal Console outside a Drupal site root

You can run Drupal Console from any directory on your system by using the `--root` option to define the Drupal root to be use in the command execution.

```
drupal --root=/var/www/drupal8.dev cr all
```

NOTE: Possible messages when executing Drupal Console outside a Drupal site root and no `--root` option provided.

When running the project outside of a Drupal 8 site root, the following message will be shown.

> In order to list all of the available commands, you should run this inside a drupal root directory.

When running the project within of a Drupal 8 site root, but site is not yet installed, the following message will be shown.

> In order to list all of the available commands you should install drupal first.

If you already have an existing Drupal 8 site and have installed the global Launcher using the instructions in 2.2, you will still need to install it into the Drupal site using instructions at 2.1 section.

How to copy configuration files

The first task you should do after installing Drupal Console is to execute the `init` command. Executing this command will copy the project configurations files to your local computer. Overriding values on these copied files is how you can change DrupalConsole behaviour.

```
drupal init [--override]
```

The first question you will be asked by the interactive mode of this command will be `Select destination to copy configuration:` and the options shown will vary depending the directory where you ran this command:

Running this command outside of a drupal site directory the options will be.

```
Select destination to copy configuration:
 [0] /etc/console/
 [1] /Users/username/.console/
>
```

Running this command within a drupal site.

```
Select destination to copy configuration:
 [0] /etc/console/
 [1] /Users/username/.console/
 [2] /path/to/drupal8.dev
>
```

The rest of the options for this command are the same regardless of the directory you execute this command.

How to download, install and serve Drupal 8

The easiest way to try Drupal 8 in your local machine is by executing the `quick:start` command.

```
drupal quick:start
```

> NOTE: You must execute `drupal init` before in order to copy the `~/.console/chain/quick-start.yml` on your local system.

The `chain` command helps you to automate command execution, allowing you to define an external YAML file containing the definition name, option and arguments of several commands and execute that list based on the sequence defined in the file.

The content of the provided `~/.console/chain/quick-start.yml` file is:

```
# How to use
# quick:start --directory="/path/to/drupal-project/"
# quick:start --directory="/path/to/drupal-project/" --profile="minimal"
# quick:start --repository="acquia/lightning-project:^8.1" --directory="/path/to/drupal-p
command:
  name: quick:start
  description: 'Download, install and serve a new Drupal project'
vars:
  repository:
    - drupal-composer/drupal-project:8.x-dev
    - acquia/lightning-project
    - acquia/reservoir-project
  profile: standard
commands:
  # Create Drupal project using DrupalComposer
  - command: exec
    arguments:
      bin: composer create-project %{{repository}} %{{directory}} --prefer-dist --no-prog
  # Install Drupal
  - command: exec
    arguments:
      bin: drupal site:install %{{profile}} --root=%{{directory}} --db-type="sqlite" --no
  # Start PHP built-in server
  - command: exec
    arguments:
      bin: drupal server --root=%{{directory}}ß
```

The previous configuration will execute several commands, in this case commands that will download and install Drupal using SQLite, and finally start the PHP's built in server, now you only need to open your browser and point it to 127.0.0.1:8088.

You can duplicate or make changes on the provided YAML file, to add commands for download modules `module:download`, install modules `module:install`, import configurations `config:import` and restore your database `database:restore` or any other command provided by DrupalConsole or a custom command by your own module.

How to use Drupal Console in a multi-site installation

Drupal Console provides support for Drupal multi-site installations. This project provides the `debug:multisite` command to debug multi-site installations and the `--uri` option to interact with multi-site installations.

How to list all known multi sites

Drupal Console uses the sites/sites.php file to determine the multi site configuration. See sites/example.site.php how to configure this file.

```
drupal debug:multisite

+---------------------+---------------------------------+
| Site                | Directory                       |
+---------------------+---------------------------------+
| drupal8.dev         | /var/www/drupal8.dev/default    |
| drupal8.multi.dev   | /var/www/drupal8.dev/multi.dev |
+---------------------+---------------------------------+

 Sites are written using the format: <port>.<domain>.<path>
```

How to execute a command against a multi-site installation

```
drupal --uri=http://drupal8.multi.dev cr all
drupal --uri=drupal8.multi.dev cr all
```

Using site alias

Drupal Console allows you to run commands from your local server but actually execute them on a local, remote or virtual (VM, Docker) Drupal installation by using site aliases.

Site alias files uses YAML format to provide a collection of predefined options once an alias is defined you can call them using a short name.

A site alias could be defined for a remote site by site using `type: ssh`. In this case the ssh command will be used to execute the command on the remote installation.

Site alias files are discoverable from the following paths:

- **Globally**: `~/.console/sites/`
- **Per site**: `/path/to/site/console/sites/`

Site alias valid options

List of valid key/value options for site alias file configuration.

- **root**: Drupal root project directory.
- **host**: Domain name of the remote system. Not required on local sites.
- **port**: The port to use when connecting via shh. The port 22 used by default.
- **user**: The username to use when connecting via ssh.
- **options**: Array of valid DrupalConsole options.
- **arguments**: Array of valid DrupalConsole arguments.
- **extra-options**: Used only when the target requires extra options, such as alternative authentication method and/or alternative identity file.
- **type**: Type of site to interact with. Allowed options `local`, `ssh`, `container`. The `local` option is used by default.

NOTE:: The values `root` and `type` are required.

Setting up your local machine

Using a site alias requires some local configuration.

Global configuration

Global configuration can provided using the copied file `~/.console/config.yml`. This information is grouped within the `remote` key.

```
application:
  ...
  remote:
    user: drupal
    port: 22
    options:
    arguments:
    type: ssh
```

Specific site configuration

Site alias configuration could be provided by adding a YAML file at `/path/to/site/console/sites/sample.yml` or `~/.console/sites/sample.yml`

```
local:
  root: /var/www/drupal8.dev
  type: local
dev:
  root: /var/www/html/drupal
  host: 140.211.10.62
  user: drupal
  type: ssh
prod:
  root: /var/www/html/docroot
  host: live.drupal.org
  user: drupal
  type: ssh
```

Debug sites.

All known sites alias can be listed by executing the `debug:site` command.

```
drupal debug:site

+--------------------+-----------------+------------------------+
| Site               | Host            | Root                   |
+--------------------+-----------------+------------------------+
| sample.local       | local           | /var/www/drupal8.dev   |
| sample.dev         | 140.211.10.62   | /var/www/html/drupal   |
| sample.prod        | live.drupal.org | /var/www/html/docroot  |
+--------------------+-----------------+------------------------+
```

A site configuration details can be shown by passing the site name as argument to the `debug:site` command.

```
drupal debug:site sample.dev

user: drupal
port: 22
options:
arguments:
root: /var/www/html/drupal
host: 140.211.10.62
type: ssh
```

NOTE: As you may notice the global configuration and the specific site configuration are merged when debugging a site. You can override any global configuration by adding those keys on the site specific configuration.

How to use Drupal Console in a remote installation

Site aliases can be executed using the `--target` option and passing the site name you want to interact with.

```
drupal --target=sample.dev cr all
```

Site aliases can also be executed by using the legacy `@` identifier as:

```
drupal @sample.dev cr all
```

Connecting to a virtual environment

You can connect to a virtual-machine/docker by providing the proper values for `extra-options` and `type`.

DrupalVM Example

```
dev:
  root: /var/www/drupalvm/drupal
  host: 192.168.88.88
  user: vagrant
  extra-options: '-o PasswordAuthentication=no -i ~/.vagrant.d/insecure_private_key'
  type: ssh
```

Drupal4Docker example

```
dev:
  root: /var/www/html
  extra-options: docker-compose exec --user=82 php
  type: container
```

When connecting using `type: container` there is no need to provide `host` and `user` values.

Creating Custom Commands

The easiest way to create a custom Command Class is to execute the `generate:command` command.

Executing the command using the interactive command questions:

```
 // Welcome to the Drupal Command generator
 Enter the extension name [config_update]:
 > example

 Enter the Command name. [example:default]:
 >

 Enter the Command Class. (Must end with the word 'Command'). [DefaultCommand]:
 >

 Is the command aware of the drupal site installation when executed?. (yes/no) [no]:
 >

 Do you want to load services from the container (yes/no) [no]:
 > yes

Type the service name or use keyup or keydown.
This is optional, press enter to continue

 Enter your service [ ]:
 > entity_type.manager
 Enter your service [ ]:
 >

 Do you confirm generation? (yes/no) [yes]:
 >

Generated or updated files

 1 - modules/custom/example/src/Command/DefaultCommand.php
 2 - modules/custom/example/console.services.yml
 3 - modules/custom/example/console/translations/en/example.default.yml
```

Executing the `generate:command` passing inline options, make sure you adjust the following command based on your requirements.

```
drupal generate:command \
--extension="example" \
--extension-type="module" \
--class="DefaultCommand" \
--name="example:default" \
--services='entity_type.manager' \
--no-interaction
```

This command execution will generate a new Command class and the service registration containing the boiler-plate required to register a new command.

The Command Class

Custom Commands should extend any of the base Classes provided by the DrupalConsole Core project.

Extending the Base `Command` class

By extending the provided Base `Command` Class, your command will be able to take advantage of the multi-language feature provided by DrupalConsole.

1.- Import the `Command` Class.

```
use Drupal\Console\Core\Command\Command;
```

2.- Extend the imported `Command` Class.

```
class DefaultCommand extends Command
```

Extending the `ContainerAwareCommand` Class.

By extending the `ContainerAwareCommand` Class on your class for the command (instead of the more basic `Command`), you also have access to the service container.

In other words, you can access to any configured Drupal service using the provided `get` method.

1.- Import the `ContainerAwareCommand` Class.

```
use Drupal\Console\Core\Command\ContainerAwareCommand;
```

2.- Extend the imported `ContainerAwareCommand` Class.

```
class DefaultCommand extends ContainerAwareCommand
```

Registering Commands

To make the console commands available within a Drupal installation, you will need to register your Command Class as a service using the `console.services.yml` file at the root of your extension (module, theme, profile) and tag service definition as `drupal.command`.

```
services:
  example.example_default:
    class: Drupal\example\Command\DefaultCommand`
    arguments: ['@entity_type.manager']
    tags:
      - { name: drupal.command }
```

> NOTE: The `arguments` section on the service definition is optional and used only if you will be injecting services from the container on your command.

Configuring the command.

You must provide a `configure` method containing the configuration of the command as name, arguments, options, etc.

```
    /**
     * {@inheritdoc}
     */
    protected function configure()
    {
        $this
            ->setName('user:login:url')
            ->setDescription($this->trans('commands.user.login.url.description'))
            ->addArgument(
                'uid',
                InputArgument::REQUIRED,
                $this->trans('commands.user.login.url.options.uid'),
                null
            );
    }
```

Command Lifecycle

Commands have three lifecycle methods:

The initialize method (optional)

This method is executed before the `interact` and `execute` methods. Its main purpose is to initialize variables used in the rest of the command methods.

The interact method (optional)

This method is executed after `initialize` and before `execute` methods. Its purpose is to 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.

The execute method (required)

This method is executed after `interact` and `initialize` methods. It contains the logic you want the command to execute.

Getting Services from the Service Container

You can access services from the service container by:

Injecting services to the command Class

Using the `arguments` section when registering your Command class at the `console.services.yml` file.

```
services:
  example.example_default:
    class: Drupal\example\Command\DefaultCommand`
    arguments: ['@entity_type.manager']
    tags:
      - { name: drupal.command }
```

Adding a new protected property.

```
  /**
   * The $entityTypeManager definition.
   *
   * @var EntityTypeManager
   */
  protected $entityTypeManager;
```

Passing the service using the `__construct` method.

```
  /**
   * Constructs a new DefaultCommand object.
   */
  public function __construct(EntityTypeManager $entity_type_manager) {
    $this->entityTypeManager = $entity_type_manager;
    parent::__construct();
  }
```

Extending the `ContainerAwareCommand` base Class on your Command class.

By doing this you have access to the service container, in other words, you have access to any configured service using the provided `get` method.

```
protected function execute(InputInterface $input, OutputInterface $output)
{
    $uid = $input->getArgument('uid');
    $entityTypeManager = $this->get('entity_type.manager');;
    if ($entityTypeManager) {
        $user = $entityTypeManager->getStorage('user')->load($uid);
    }
}
```

Generating Code with a Command

To generate code from a Command, there are three steps:

1. Create a Twig template for the code you want to generate.
2. Create and register a custom Generator service for rendering the Twig template.
3. Inject the custom Generator service into your Command.

Creating a Twig Template for the Generated Code

All Twig templates should be placed inside a `templates` subdirectory of your module or extension. For example:

`templates/module/info.yml.twig`

```
name: {{ module }}
type: {{ type }}
description: {{ description }}
core: {{ core }}
package: {{ package }}
{% if dependencies %}
dependencies:
{% for dependency in dependencies %}
  - {{ dependency }}
{% endfor %}
{% endif %}
```

Creating a Custom Generator Service

To create a custom Generator service, first create a class that extends from `Drupal\Console\Core\Generator\Generator`. For example:

`src/Generator/ModuleGenerator`

```
namespace Drupal\your_extension\Generator;

use Drupal\Console\Core\Generator\Generator;

class ModuleGenerator extends Generator
{
  ...
}
```

Use the `renderFile()` method inherited from the `Generator` class to render your Twig template.

```
public function generate($module, $machineName, $output_dir, $description, $core, $packag

  $parameters = [
    'module' => $module,
    'core' => $core,
    'description' => $description,
    'package' => $package,
    'dependencies' => $dependencies,
  ];

  $this->renderFile(
    'module/info.yml.twig',
    $output_dir.'/'.$machineName.'.info.yml',
    $parameters
  );

}
```

Finally, register your Generator class as a custom service in `console.services.yml`.

```
services:
  your_extension.module_generator:
    class: Drupal\your_extension\Generator\ModuleGenerator
    tags:
      - { name: drupal.generator }
```

Be sure to include the `drupal.generator` tag so that the Twig renderer is properly initialized.

Injecting Your Custom Generator Into Your Command

In `console.services.yml`, add your custom Generator service as an argument for your custom Command.

```
services:
  your_extension.generate_module:
    class: Drupal\your_extension\Command\Generate\ModuleCommand
    arguments: ['@your_extension.module_generator']
    tags:
       - { name: drupal.command }
```

And add your custom Generator to the constructor parameters for your Command class:

```
use Drupal\your_extension\Generator\ModuleGenerator;

class ModuleCommand extends Command
{
  /**
   * ModuleCommand constructor.
   *
   * @param \Drupal\your_extension\Generator\ModuleGenerator $generator
   */
  public function __construct(ModuleGenerator $generator) {
    $this->generator = $generator;
    parent::__construct();
  }

  ...
}
```

You can now call your Generator from the `execute()` method in your Command to output rendered code files:

```
protected function execute(InputInterface $input, OutputInterface $output) {

  ...

  $this->generator->generate(
    $module,
    $machineName,
    $modulePath,
    $description,
    $core,
    $package,
    $dependencies
  );

}
```

Contributing new features

If you create a new custom command or something else which would be useful for *other* Drupal installations, please consider forking the Drupal Console project on GitHub. Then just create an issue for a "feature request", put your work in a new Git branch on your fork, publish your branch and add a pull request on GitHub (including your issue ID in the PR title). The Drupal Console maintainers are very happy to accept useful contributions and usually do so quite promptly.

Getting support with becoming a contributor

If you want to contribute to Drupal Console and have any difficulty getting oriented to the process, you can find Instant Messaging support on Gitter IM.

Standard GitHub Workflow

If you haven't yet contributed to a project on GitHub, or are not sure you remember the workflow, you might first want to read the documentation about pull requests. You may also wish to download the GitHub application (Mac | Windows, which simplifies the workflow a bit and provides a nice GUI for your contributions).

Github-flavored Markdown

This documentation is written in “Github-flavored Markdown”, which is rendered on Github, directly, as HTML. If you are *already* familiar with Markdown, GFL only adds a few useful tweaks, otherwise you may want to read Github’s Markdown Basics primer.

Project requirements

Download Git

We recommend downloading Git from http://git-scm.com/downloads

Download Composer

Run this in your terminal to get the latest Composer version:

```
curl -sS https://getcomposer.org/installer | php
```

Or if you don't have curl:

```
php -r "readfile('https://getcomposer.org/installer');" | php
```

This installer script will simply check some php.ini settings, warn you if they are set incorrectly, and then download the latest composer.phar in the current directory

You can run this terminal command to make Composer easily accessible, from anywhere on your system:

```
mv composer.phar /usr/local/bin/composer
```

Getting the project

The Drupal Console is a modular project using multiple repositories.

Main repositories:

- drupal/console
- drupal/console-core
- drupal/console-extend-plugin
- drupal-console-dotenv

Additional projects:

- drupal/console-develop
- drupal-console-yaml

Languages are also managed into separated repositories:

- drupal-console-en
- drupal-console-es

Fork

You should fork the repositories that you want to contribute. For this task you can use the github GUI.

Clone

You need to define a directory where you will clone the repositories. i.e. `/Users/username/drupal-console-code`

```
cd /Users/username/drupal-console-code
git clone git@github.com:[your-github-user-here]/drupal-console.git
git clone git@github.com:[your-github-user-here]/drupal-console-core.git
git clone git@github.com:[your-github-user-here]/drupal-console-en.git
```

Install dependencies

Now that you have cloned the repositories, you need to download dependencies using Composer.

```
cd /Users/username/drupal-console-code/[cloned-repository]
composer install
```

Read the next step to learn how to link and test these repositories into a Drupal site.

Running the project

In order to contribute you will need a working Drupal site linked to the packages you cloned and forked.

Execute the automated process

DrupalConsole provides you with a command to take care of all this process for you.

```
drupal develop:contribute \
--drupal=/path/to/drupal8.dev \
--code=/Users/username/drupal-console-code/
```

The option `--drupal` is the directory where the new drupal site will be created and the option `--code` is the parent directory where the different DrupalConsole repositories were cloned.

> NOTES:
>
> You must execute `drupal init` before in order to copy the `~/.console/chain/develop-contribute.yml` on your local system.
>
> Make sure you have the latest version of DrupalConsole. Get the latest version of DrupalConsole by following the instructions as mentioned here.

Execute all the steps manually

If you want to execute all the steps manually you can follow the instructions below:

Download Drupal and DrupalConsole

```
composer create-project \
drupal-composer/drupal-project:8.x-dev \
drupal8.dev \
--prefer-dist \
--no-progress \
--no-interaction
```

Install Drupal using SQLite

```
drupal site:install standard --db-type="sqlite" --no-interaction
```

> NOTE: You can install drupal using MySQL by executing the `site:install` command and answering the questions from the interactive mode or passing the required options.

Download the Drupal Console Develop package

```
composer require drupal/console-develop --dev
```

Create a symbolic link between Drupal and forked repositories

```
drupal develop:create:symlinks \
--code-directory=/Users/username/drupal-console-code/
```

Downloading additional Drupal Console language or packages

If you want to contribute translating Drupal Console to Spanish you should:

1.- Download it on the drupal site by executing the following command.

```
composer require drupal/console-es
```

2.- Fork and clone the repo to your local directory.

3.- Execute the `develop:create:symlinks` again, to create symlinks including the recently added package.

This applies for additional languages and packages i.e drupal/console-yaml.

Wrapping up

Now you can do the required changes and start contributing, commit you changes, push code to your forked repositories and create a Pull Request to the respective repository.

Happy coding ... thanks for contributing to Drupal Console.

Keeping your fork up to date

After some time your forked repository and the original one (called upstream) will eventually get out of sync leaving you with an old, unsupported version.

To sync changes you make in a fork with the original repository, you should:

Configuring a remote fork:

Specify and configure a new remote upstream repository that points to the upstream repository in Git.

```
git remote add upstream https://github.com/hechoendrupal/drupal-console.git
```

For detailed information please visit Github's guide Configuring a remote fork

Syncing your fork

Sync your fork to keep it up-to-date with the upstream repository.

```
git fetch upstream
git merge upstream/master
```

For detailed information please visit Github's guide Syncing a fork

Creating an Issue

If you found an issue or maybe you like to propose a new feature to the project, you need to access the following link:

https://github.com/hechoendrupal/drupal-console/issues/new

Please review the guidelines for contributing to Drupal Console at:

https://github.com/hechoendrupal/drupal-console/blob/master/CONTRIBUTING.md

Creating a Pull Request

Remember to create a branch before start developing! It's name should contain the issue id and a slug to tell what the thing you're working on is about, for example: `1337-lorem-ipsum`

If you haven't yet contributed to a project on GitHub, or aren't still sure what the workflow looks like, read the documentation about pull requests. You may also wish to download the GitHub application (Mac | Windows, which simplifies the workflow a bit and provides a nice GUI for your contributions).

Contribute to this documentation

This documentation is a work-in-progress and *you are welcome to pitch in and help*. Simply fork the Drupal Console Book on GitHub. If you haven't yet contributed to a project on GitHub, or aren't still sure what the workflow looks like, read the documentation about pull requests. You may also wish to download the GitHub application (Mac | Windows, which simplifies the workflow a bit and provides a nice GUI for your contributions).

Available Drupal Console Commands

Note: Drupal Console commands *must* be run from the root of a Drupal 8 installation.

Drupal Console Command	Details
misc	
about	Displays basic information about Drupal Console project
chain	Chain command execution
check	System requirement checker
exec	Execute an external command.
help	Displays help for a command
init	Copy configuration files.
list	Lists all available commands
shell	Open a shell providing an interactive REPL (Read–Eval–Print-Loop).
server	Runs PHP built-in web server
cache	
cache:rebuild	Rebuild and clear all site caches.
config	
config:delete	Delete configuration
config:diff	Output configuration items that are different in active configuration compared with a directory.
config:edit	Change a configuration object with a text editor.
config:export	Export current application configuration.
config:export:content:type	Export a specific content type and their fields.
config:export:single	Export a single configuration or a list of configurations as yml file(s).
config:export:view	Export a view in YAML format inside a provided module to reuse in other website.
config:import	Import configuration to current application.
config:import:single	Import a single configuration or a list of configurations.

config:override	Override config value in active configuration.
config:validate	Validate a drupal config against its schema
create	
create:comments	Create dummy comments for your Drupal 8 application.
create:nodes	Create dummy nodes for your Drupal 8 application.
create:terms	Create dummy terms for your Drupal 8 application.
create:users	Create dummy users for your Drupal 8 application.
create:vocabularies	Create dummy vocabularies for your Drupal 8 application.
cron	
cron:execute	Execute cron implementations by module or execute all crons
cron:release	Release cron system lock to run cron again
database	
database:add	Add a database to settings.php
database:client	Launch a DB client if it's available
database:connect	Shows DB connection
database:drop	Drop all tables in a given database.
database:dump	Dump structure and contents of a database
database:log:clear	Remove events from DBLog table, filters are available
database:log:poll	Poll the watchdog and print new log entries every x seconds
database:query	Executes a SQL statement directly as argument
database:restore	Restore structure and contents of a database.
debug	
debug:breakpoints	Displays breakpoints available in application
debug:cache:context	Displays current cache context for the application.
debug:chain	List available chain files.
debug:config	List configuration objects names and single configuration object.
debug:config:settings	Displays current key:value on settings file.
debug:config:validate	Validate a schema implementation before a module

	is installed.
debug:container	Displays current services for an application.
debug:cron	List of modules implementing a cron
debug:database:log	Displays current log events for the application
debug:database:table	Show all tables in a given database.
debug:entity	Debug entities available in the system
debug:event	Displays current events
debug:features	List registered features.
debug:image:styles	List image styles on the site
debug:libraries	Displays libraries available in application
debug:migrate	Display current migration available for the application
debug:module	Displays current modules available for application
debug:multisite	List all multisites available in system
debug:permission	Displays all user permissions.
debug:plugin	Displays all plugin types.
debug:queue	Displays the queues of your application
debug:rest	Display current rest resource for the application
debug:router	Displays current routes for the application or information for a particular route
debug:settings	List user Drupal Console settings.
debug:site	List all known local and remote sites.
debug:state	Show the current State keys.
debug:test	List Test Units available for the application.
debug:theme	Displays current themes for the application
debug:update	Displays current updates available for the application
debug:user	Displays current users for the application
debug:views	Displays current views resources for the application
debug:views:plugins	Displays current views plugins for the application
devel	
devel:dumper	Change the devel dumper plugin
develop	

develop:contribute	
develop:example	
develop:example:container:aware	
develop:gitbook	Update gitbook
dotenv	
dotenv:debug	Debug Dotenv debug values.
dotenv:init	Dotenv initializer.
entity	
entity:delete	Delete an specific entity
extend	
extend:example:one	Drupal Console extend example
extend:example:two	Drupal Console extend example
features	
features:import	Import module config.
field	
field:info	View information about fields.
generate	
generate:authentication:provider	Generate an Authentication Provider
generate:breakpoint	Generate breakpoint
generate:cache:context	Generate a cache context
generate:command	Generate commands for the console.
generate:controller	Generate & Register a controller
generate:doc:cheatsheet	commands.generate.doc.cheatsheet.description
generate:doc:dash	commands.generate.doc.dash.description
generate:doc:data	commands.generate.doc.data.description
generate:doc:gitbook	commands.generate.doc.gitbook.description
generate:entity:bundle	Generate a new content type (node / entity bundle)
generate:entity:config	Generate a new config entity
generate:entity:content	Generate a new content entity
generate:event:subscriber	Generate an event subscriber
generate:form	Generate a new "%s"
generate:form:alter	Generate an implementation of hook_form_alter()

generate:form:alter	or hook_form_FORM_ID_alter
generate:form:config	commands.generate.form.description
generate:help	Generate an implementation of hook_help()
generate:module	Generate a module.
generate:module:file	Generate a .module file
generate:permissions	commands.generate.permission.description
generate:plugin:block	Generate a plugin block
generate:plugin:ckeditorbutton	Generate CKEditor button plugin.
generate:plugin:condition	Generate a plugin condition.
generate:plugin:field	Generate field type, widget and formatter plugins.
generate:plugin:fieldformatter	Generate field formatter plugin.
generate:plugin:fieldtype	Generate field type plugin.
generate:plugin:fieldwidget	Generate field widget plugin.
generate:plugin:imageeffect	Generate image effect plugin.
generate:plugin:imageformatter	Generate image formatter plugin.
generate:plugin:mail	Generate a plugin mail
generate:plugin:migrate:process	Generate a migrate process plugin
generate:plugin:migrate:source	Generate a migrate source plugin
generate:plugin:rest:resource	Generate plugin rest resource
generate:plugin:rulesaction	Generate a plugin rule action
generate:plugin:skeleton	Generate an implementation of a skeleton plugin
generate:plugin:type:annotation	Generate a plugin type with annotation discovery
generate:plugin:type:yaml	Generate a plugin type with Yaml discovery
generate:plugin:views:field	Generate a custom plugin view field.
generate:post:update	commands.generate.post:update.description
generate:profile	Generate a profile.
generate:routesubscriber	Generate a RouteSubscriber
generate:service	Generate service
generate:theme	Generate a theme.
generate:twig:extension	Generate a Twig extension.
generate:update	Generate an implementation of hook_update_N()

image	
image:styles:flush	Execute flush function by image style or execute all flush images styles
locale	
locale:language:add	Add a language to be supported by your site
locale:language:delete	Delete a language to be supported by your site
locale:translation:status	List available translation updates
migrate	
migrate:execute	Execute a migration available for application
migrate:rollback	Rollback one or multiple migrations
migrate:setup	Load and create the relevant migrations for a provided legacy database
module	
module:dependency:install	commands.module.install.dependencies.description
module:download	Download module or modules in application
module:install	Install module or modules in the application
module:path	Returns the relative path to the module (or absolute path)
module:uninstall	Uninstall module or modules in the application
module:update	Update core, module or modules in the application
multisite	
multisite:new	Sets up the files for a new multisite install.
node	
node:access:rebuild	Rebuild node access permissions.
queue	
queue:run	Process the selected queue.
quick	
quick:start	Download, install and serve a new Drupal project
rest	
rest:disable	Disable a rest resource for the application
rest:enable	Enable a rest resource for the application
router	
router:rebuild	Rebuild routes for the application

sample	
sample:default	commands.sample.default.description
settings	
settings:set	Change a specific setting value in DrupalConsole config file
site	
site:import:local	Import/Configure an existing local Drupal project
site:install	Install a Drupal project
site:maintenance	Switch site into maintenance mode
site:mode	Switch system performance configuration
site:new	Download a new Drupal project
site:statistics	Show the current statistics of website.
site:status	View current Drupal Installation status
state	
state:delete	Delete State
state:override	Override a State key.
taxonomy	
taxonomy:term:delete	Delete taxonomy terms from a vocabulary
test	
test:run	Run Test unit from tests available for application
theme	
theme:download	Download theme in application
theme:install	Install theme or themes in the application
theme:path	Returns the relative path to the theme (or absolute path)
theme:uninstall	Uninstall theme or themes in the application
translation	
translation:cleanup	commands.translation.cleanup.description
translation:pending	commands.translation.pending.description
translation:stats	commands.translation.stats.description
translation:sync	commands.translation.sync.description
update	

update:entities	Applying Entity Updates
update:execute	Execute a specific Update N function in a module, or execute all
user	
user:create	Create users for the application
user:delete	Delete users for the application
user:login:clear:attempts	Clear failed login attempts for an account.
user:login:url	Returns a one-time user login url.
user:password:hash	Generate a hash from a plaintext password.
user:password:reset	Reset password for a specific user.
user:role	Adds/removes a role for a given user
views	
views:disable	Disable a View
views:enable	Enable a View

Available options

Option	Details
--help	Display this help message
--quiet	Suppress all output from the command
--verbose	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output, and 3 for debug
--version	Display this application version
--ansi	Force ANSI output
--no-ansi	Disable ANSI output
--no-interaction	Do not ask any interactive question
--env	The Environment name
--root	Define the Drupal root to be used in command execution
--debug	application.options.debug
--learning	Generate a verbose code output
--generate-chain	Shows command options and arguments as yaml output to be used in chain command
--generate-inline	Shows command options and arguments as inline command
--generate-doc	Shows command options and arguments as markdown
--target	Site name you want to interact with (for local or remote sites)
--uri	URI of the Drupal site to use (for multi-site environments or when running on an alternate port)
--yes	Skip confirmation and proceed

Available arguments

Argument	Details
command	The command to execute

about

Displays basic information about Drupal Console project

Usage:

```
drupal about
```

chain

Chain command execution

Usage:

```
drupal chain [arguments] [options]
```

Available options

Option	Details
--file	User defined file containing commands to get executed.
--placeholder	commands.chain.options.placeholder
--help	Display this help message
--quiet	Do not output any message
--verbose	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
--version	Display this application version
--ansi	Force ANSI output
--no-ansi	Disable ANSI output
--no-interaction	Do not ask any interactive question
--env	The Environment name
--root	Define the Drupal root to be used in command execution
--debug	application.options.debug
--learning	Generate a verbose code output
--generate-chain	Shows command options and arguments as yaml output to be used in chain command
--generate-inline	Shows command options and arguments as inline command
--generate-doc	Shows command options and arguments as markdown
--target	Site name you want to interact with (for local or remote sites)
--uri	URI of the Drupal site to use (for multi-site environments or when running on an alternate port)
--yes	Skip confirmation and proceed

Available arguments

Argument	Details
command	The command to execute

Examples

- Providing a file option using full path.

```
drupal chain \
--file="/path/to/file/chain-file.yml"
```

check

System requirement checker

Usage:

```
drupal check
```

exec

Execute an external command.

Usage:

```
drupal exec [arguments] [options]
```

Available options

Option	Details
--working-directory	The current working directory.

Available arguments

Argument	Details
bin	Executable command.

help

Displays help for a command

Usage:

```
drupal help [arguments] [options]
```

Available options

Option	Details
--xml	To output list as XML
--raw	To output raw command list
--format	The output format (txt, xml, json, or md)

Available arguments

Argument	Details
command_name	The command name

init

Copy configuration files.

Usage:

```
drupal init [options]
```

Available options

Option	Details
--destination	Destination directory to copy files
--override	Override configurations files flag
--autocomplete	Autocomplete tool files flag.

list

Lists all available commands

Usage:

```
drupal list [arguments] [options]
```

Available options

Option	Details
--xml	To output list as XML
--raw	To output raw command list
--format	The output format (txt, xml, json, or md)

Available arguments

Argument	Details
namespace	The namespace name

shell

Open a shell providing an interactive REPL (Read–Eval–Print-Loop).

Usage:

```
drupal shell
```

server

Runs PHP built-in web server

Usage:

```
drupal server [arguments]
```

Available arguments

Argument	Details
address	The address:port values

Examples

- Run using default address argument value 127.0.0.1:8088

  ```
  drupal server
  ```

- Passing address argument to use a different port number

  ```
  drupal server 127.0.0.1:8089
  ```

- Running default address argument values, using --root option to define the Drupal root

  ```
  drupal --root=/var/www/drupal8.dev server
  ```

cache:rebuild

Rebuild and clear all site caches.

Usage:

```
drupal cache:rebuild [arguments]
cr
```

Available arguments

Argument	Details
cache	Only clear a specific cache.

Examples

- Rebuild all caches

  ```
  drupal cr all
  ```

- Rebuild discovery cache

  ```
  drupal cr discovery
  ```

config:delete

Delete configuration

Usage:

```
drupal config:delete [arguments]
cd
```

Available arguments

Argument	Details
type	Configuration type.
name	Configuration name.

Examples

- Provide a config type and a config name

  ```
  drupal config:delete active all
  ```

config:diff

Output configuration items that are different in active configuration compared with a directory.

Usage:

```
drupal config:diff [arguments] [options]
cdi
```

Available options

Option	Details
--reverse	See the changes in reverse (i.e diff a directory to the active configuration).

Available arguments

Argument	Details
directory	The directory to diff against. If omitted, choose from Drupal config directories.

Examples

- Provide a config directory

  ```
  drupal config:diff ../config/path
  ```

config:edit

Change a configuration object with a text editor.

Usage:

```
drupal config:edit [arguments]
ced
cdit
```

Available arguments

Argument	Details
config-name	Configuration object name, for example "user.settings".
editor	Editor, for example "vim" or "gedit".

Examples

- Edit system cron configurations with "vim" (default editor).

  ```
  drupal config:edit system.cron
  ```

- Edit system cron configurations with "gedit".

  ```
  drupal config:edit system.cron gedit
  ```

config:export

Export current application configuration.

Usage:

```
drupal config:export [options]
ce
```

Available options

Option	Details
--directory	Define the export directory to save the configuration output.
--tar	If set, the configuration will be exported to an archive file.
--remove-uuid	If set, the configuration will be exported without uuid key.
--remove-config-hash	If set, the configuration will be exported without the default site hash key.

Examples

- Optional you can add the path to export

```
drupal config:export  \
--directory="path/to/export"
```

- If export will be in a compressed file and/or if uuid and config hashes will be removed.

```
drupal config:export  \
--directory="path/to/export" \
--tar \
--remove-uuid \
--remove-config-hash
```

config:export:content:type

Export a specific content type and their fields.

Usage:

```
drupal config:export:content:type [arguments] [options]
cect
```

Available options

Option	Details
--module	The Module name.
--optional-config	Export content type as an optional YAML configuration in your module

Available arguments

Argument	Details
content-type	Content Type to be exported

Examples

- Provide a content type and module name

```
drupal config:export:content:type page \
--module="demo"
```

- If you want export content type provide the optional config

```
drupal config:export:content:type page \
--module="demo" \
--optional-config
```

config:export:single

Export a single configuration or a list of configurations as yml file(s).

Usage:

```
drupal config:export:single [options]
ces
```

Available options

Option	Details
--name	commands.config.export.single.options.name
--directory	commands.config.export.arguments.directory
--module	The Module name.
--include-dependencies	Export dependencies of the configuration as well.
--optional	Export config as an optional YAML configuration in your module
--remove-uuid	If set, the configuration will be exported without uuid key.
--remove-config-hash	If set, the configuration will be exported without the default site hash key.

Examples

- Provide config settings name to be exported

  ```
  drupal config:export:single \
  --name=config.settings.name
  ```

- if uuid and/or config hashes will be removed.

  ```
  drupal config:export:single \
  --name=config.settings.name \
  --remove-uuid \
  --remove-config-hash
  ```

config:export:view

Export a view in YAML format inside a provided module to reuse in other website.

Usage:

```
drupal config:export:view [arguments] [options]
cev
```

Available options

Option	Details
--module	The Module name.
--optional-config	Export view as an optional YAML configuration in your module
--include-module-dependencies	Include module dependencies in module info YAML file

Available arguments

Argument	Details
view-id	View ID

Examples

- Provide a view id

  ```
  drupal config:export:view viewid
  ```

- You can provide the interactive values like parameter.

  ```
  drupal config:export:view viewid \
  --module="modulename" \
  --optional-config \
  --include-module-dependencies
  ```

config:import

Import configuration to current application.

Usage:

```
drupal config:import [options]
ci
```

Available options

Option	Details
--file	Path to an archive file of configuration to import.
--directory	Path to a directory of configuration to import.
--remove-files	Remove files after synchronization.

Examples

- Provide a configuration file

  ```
  drupal config:import \
  --file=/path/to/config/file
  ```

- Provide a configuration directory

  ```
  drupal config:import  \
  --directory=/path/to/config/dir
  ```

config:import:single

Import a single configuration or a list of configurations.

Usage:

```
drupal config:import:single [options]
cis
```

Available options

Option	Details
--file	The file(s) name or file(s) absolute path to import
--directory	commands.config.import.arguments.directory

Examples

- Providing a file option using full path.

  ```
  drupal config:import:single \
  --file="/path/to/file/block.block.default_block.yml"
  ```

- Providing file and directory options

  ```
  drupal config:import:single  \
  --file="block.block.default_block.yml" \
  --directory="/path/to/directory"
  ```

config:override

Override config value in active configuration.

Usage:

```
drupal config:override [arguments]
co
```

Available arguments

Argument	Details
name	Configuration name
key	Key
value	Value

Examples

- Set the Contact module flood limit to 10.

  ```
  drupal config:override contact.settings flood.limit 10
  ```

config:validate

Validate a drupal config against its schema

Usage:

```
drupal config:validate [arguments]
cv
```

Available arguments

Argument	Details
name	

Examples

- Provide the configuration name.

  ```
  drupal config:validate configuration.name
  ```

create:comments

Create dummy comments for your Drupal 8 application.

Usage:

```
drupal create:comments [arguments] [options]
crc
```

Available options

Option	Details
--limit	How many comments would you like to create
--title-words	Maximum number of words in comment titles
--time-range	How far back in time should the comments be dated

Available arguments

Argument	Details
node-id	Node ID where the comments will be created

Examples

- Provide the node id where the comments will be generated.

  ```
  drupal create:comments  node-id
  ```

- Provide number of comments to generate, max title words and time range.

  ```
  drupal create:comments  node-id \
  --limit="2" \
  --title-words="5" \
  --time-range="1"
  ```

create:nodes

Create dummy nodes for your Drupal 8 application.

Usage:

```
drupal create:nodes [arguments] [options]
crn
```

Available options

Option	Details
--limit	How many nodes would you like to create
--title-words	Maximum number of words in node titles
--time-range	How far back in time should the nodes be dated
--language	commands.create.nodes.options.language

Available arguments

Argument	Details
content-types	Content type(s) to be used in node creation

Examples

- Provide the content type name.

  ```
  drupal create:nodes content-name
  ```

- Provide the limit of publications, limit of title words, time range and language.

  ```
  drupal create:nodes content-name \
  --limit="5" \
  --title-words="5" \
  --time-range="1" \
  --language="und"
  ```

create:terms

Create dummy terms for your Drupal 8 application.

Usage:

```
drupal create:terms [arguments] [options]
crt
```

Available options

Option	Details
--limit	How many terms would you like to create
--name-words	Maximum number of words in term names

Available arguments

Argument	Details
vocabularies	Vocabulary(s) to be used in terms creation

Examples

- Provide the vocabulary term name.

  ```
  drupal create:terms vocabulary
  ```

- Provide the limit of terms to add and limit of title words.

  ```
  drupal create:terms tags \
  --limit="10" \
  --name-words="5"
  ```

create:users

Create dummy users for your Drupal 8 application.

Usage:

```
drupal create:users [arguments] [options]
cru
```

Available options

Option	Details
--limit	How many users would you like to create
--password	Password to be set to users created
--time-range	How far back in time should the users be dated

Available arguments

Argument	Details
roles	Role(s) to be used in user creation

Examples

- Provide the user role.

  ```
  drupal create:users role
  ```

- Provide the number of users to create, password and time range to create.

  ```
  drupal create:users role \
  --limit="5" \
  --password="usersnewpassword" \
  --time-range="1"
  ```

create:vocabularies

Create dummy vocabularies for your Drupal 8 application.

Usage:

```
drupal create:vocabularies [options]
crv
```

Available options

Option	Details
--limit	How many vocabularies would you like to create
--name-words	Maximum number of words in vocabulary names

Examples

- Provide the number of vocabularies to create and maximum number of words in vocabulary names

  ```
  drupal create:vocabularies \
  --limit="5" \
  --name-words="5"
  ```

cron:execute

Execute cron implementations by module or execute all crons

Usage:

```
drupal cron:execute [arguments]
croe
cre
```

Available arguments

Argument	Details
module	The Module name.

Examples

- Execute the cron globally

  ```
  drupal cron:execute
  ```

- Execute the cron on the specified module

  ```
  drupal cron:execute \
  <module>
  ```

cron:release

Release cron system lock to run cron again

Usage:

```
drupal cron:release
cror
crr
```

Examples

- Execute the cron globally

  ```
  drupal cron:execute
  ```

database:add

Add a database to settings.php

Usage:

```
drupal database:add [options]
dba
```

Available options

Option	Details
--database	The database name
--username	The database username
--password	The database password
--prefix	The database prefix
--host	The database host address
--port	The database host port
--driver	The database driver

Examples

- Add a database to the settings.php

```
drupal database:add \
--database=DATABASE \
--username=USERNAME \
--password=PASSWORD
```

database:client

Launch a DB client if it's available

Usage:

```
drupal database:client [arguments]
dbc
```

Available arguments

Argument	Details
database	Database key from settings.php

Examples

- Launch the default client or could launch another regarding the specification on the argument

  ```
  drupal database:client <database>
  ```

database:connect

Shows DB connection

Usage:

```
drupal database:connect [arguments]
dbco
```

Available arguments

Argument	Details
database	Database key from settings.php

Examples

- Connects to an specified database, or the default if not arguments passed

  ```
  drupal database:connect \
  <database>
  ```

database:drop

Drop all tables in a given database.

Usage:

```
drupal database:drop [arguments]
dbd
```

Available arguments

Argument	Details
database	Database key from settings.php

Examples

- Drop the tables on the database specified on the argument

  ```
  drupal database:drop \
  <database>
  ```

database:dump

Dump structure and contents of a database

Usage:

```
drupal database:dump [arguments] [options]
dbdu
```

Available options

Option	Details
--file	The filename for your database backup
--gz	Pass this option if you want the sql result file gzipped

Available arguments

Argument	Details
database	Database key from settings.php

Examples

- Dump default database or the one specified on the argument

  ```
  drupal database:dump \
  <database>
  ```

- Dump in gz compressed format

  ```
  drupal database:dump \
  --gz
  ```

database:log:clear

Remove events from DBLog table, filters are available

Usage:

```
drupal database:log:clear [arguments] [options]
dblc
```

Available options

Option	Details
--type	Filter events by a specific type
--severity	Filter events by a specific level of severity
--user-id	Filter events by a specific user id

Available arguments

Argument	Details
event-id	DBLog event ID

Examples

- Clear the database log from DBLog table

```
drupal database:log:clear \
<database>
```

- Clear the database log from DBLog table using filters

```
drupal database:log:clear \
<database> \
--type=TYPE \
--severity=SEVERITY
```

database:log:poll

Poll the watchdog and print new log entries every x seconds

Usage:

```
drupal database:log:poll [arguments] [options]
dblp
```

Available options

Option	Details
--type	Filter events by a specific type
--severity	Filter events by a specific level of severity
--user-id	Filter events by a specific user id

Available arguments

Argument	Details
duration	Duration in seconds which to sleep between database reads

Examples

- Print the log entries on screen every x seconds

  ```
  drupal database:log:poll \
  100
  ```

database:query

Executes a SQL statement directly as argument

Usage:

```
drupal database:query [arguments] [options]
dbq
```

Available options

Option	Details
--quick	Do not cache each query result, print each row as it is received
--debug	Prints debugging information and memory and CPU usage statistics when the program exits
--html	Produce HTML output
--xml	Produce XML output
--raw	Special characters are not escaped in the output.
--vertical	Print query output rows vertically
--batch	Print results using tab as the column separator, with each row on a new line. With this option, mysql does not use the history file

Available arguments

Argument	Details
query	The SQL statement to execute
database	Database key from settings.php

Examples

- Send a database query

  ```
  drupal database:query 'select * from node limit 0,1'
  ```

database:restore

Restore structure and contents of a database.

Usage:

```
drupal database:restore [arguments] [options]
dbr
```

Available options

Option	Details
--file	The filename for your database backup file

Available arguments

Argument	Details
database	Database key from settings.php

Examples

- Restore the database file dump to the database default or another one specified

  ```
  drupal database:restore \
  --file='/srv/dump/db.sql'
  ```

debug:breakpoints

Displays breakpoints available in application

Usage:

```
drupal debug:breakpoints [arguments]
dbre
```

Available arguments

Argument	Details
group	Enter Breakpoint Group Name

Examples

- Provide a group name.

  ```
  drupal breakpoints:debug bartik
  ```

debug:cache:context

Displays current cache context for the application.

Usage:

```
drupal debug:cache:context
dcc
```

Examples

- Displays cache context

  ```
  drupal debug:cache:context
  ```

debug:chain

List available chain files.

Usage:

```
drupal debug:chain
dch
```

debug:config

List configuration objects names and single configuration object.

Usage:

```
drupal debug:config [arguments]
dc
```

Available arguments

Argument	Details
name	Configuration object name, for example "system.site".

Examples

- List all configuration object names.

  ```
  drupal config:debug
  ```

- Display system site configurations values.

  ```
  drupal config:debug system.site
  ```

- List all system configuration names.

  ```
  drupal config:debug | grep system
  ```

debug:config:settings

Displays current key:value on settings file.

Usage:

```
drupal debug:config:settings
dcs
```

Examples

- Displays key:value as per the settings file

  ```
  drupal debug:config:settings
  ```

debug:config:validate

Validate a schema implementation before a module is installed.

Usage:

```
drupal debug:config:validate [arguments] [options]
dcv
```

Available options

Option	Details
--schema-name	

Available arguments

Argument	Details
filepath	
schema-filepath	

Examples

- Validates a schema

  ```
  drupal debug:config:validate \
  /path/to/file \
  /path/to/schema-filepath
  ```

debug:container

Displays current services for an application.

Usage:

```
drupal debug:container [arguments] [options]
dco
```

Available options

Option	Details
--parameters	Service name.

Available arguments

Argument	Details
service	Service name.
method	Method name.
arguments	Array of Arguments in CSV or JSON format.

Examples

- Displays the views.views_data_helper services

  ```
  drupal debug:container views.views_data_helper
  ```

debug:cron

List of modules implementing a cron

Usage:

```
drupal debug:cron
dcr
```

Examples

- This will show a list with modules implementing the cron hook

  ```
  drupal debug:cron
  ```

debug:database:log

Displays current log events for the application

Usage:

```
drupal debug:database:log [arguments] [options]
dbb
```

Available options

Option	Details
--type	Filter events by a specific type
--severity	Filter events by a specific level of severity
--user-id	Filter events by a specific user id
--asc	List events in ascending order
--limit	Limit results to a specific number
--offset	Starting point of a limit
--yml	Print in a yml style

Available arguments

Argument	Details
event-id	DBLog event ID

Examples

- List all the entries on the log

  ```
  drupal debug:database:log
  ```

- List specific log entry by Event ID

  ```
  drupal debug:database:log 21228
  ```

debug:database:table

Show all tables in a given database.

Usage:

```
drupal debug:database:table [arguments] [options]
ddt
```

Available options

Option	Details
--database	Database key from settings.php

Available arguments

Argument	Details
table	Table to debug

Examples

- Show all tables on a database

  ```
  drupal debug:database:table
  ```

- Show fields on the node table or another specified on the argument

  ```
  drupal debug:database:table node
  ```

debug:entity

Debug entities available in the system

Usage:

```
drupal debug:entity [arguments]
de
```

Available arguments

Argument	Details
entity-type	commands.debug.entity.arguments.entity-type

Examples

- Displays the available entities

  ```
  drupal debug:entity
  ```

debug:event

Displays current events

Usage:

```
drupal debug:event [arguments]
dev
```

Available arguments

Argument	Details
event	Event to debug

Examples

- List all the events that could be debugged

  ```
  drupal debug:evet
  ```

- Show the information for the kernel.request event

  ```
  drupal debug:event kernel.request
  ```

debug:features

List registered features.

Usage:

```
drupal debug:features [arguments]
```

Available arguments

Argument	Details
bundle	Bundle name

debug:image:styles

List image styles on the site

Usage:

```
drupal debug:image:styles
dis
```

Examples

- List all images styles on the site

  ```
  drupal debug:image:styles
  ```

debug:libraries

Displays libraries available in application

Usage:

```
drupal debug:libraries [arguments]
dl
```

Available arguments

Argument	Details
group	Enter Libraries Name

Examples

- List all available libraries

  ```
  drupal debug:libraries
  ```

- List block library information

  ```
  drupal debug:libraries block
  ```

debug:migrate

Display current migration available for the application

Usage:

```
drupal debug:migrate [arguments]
mid
```

Available arguments

Argument	Details
tag	Migrate tag

Examples

- Displays current migration

  ```
  drupal debug:migrate
  ```

debug:module

Displays current modules available for application

Usage:

```
drupal debug:module [arguments] [options]
dm
```

Available options

Option	Details	
--status	Module status [installed	uninstalled]
--type	Module type [core	no-core]

Available arguments

Argument	Details
module	Module name

Examples

- Display all installed modules

  ```
  drupal mod --status=installed
  ```

- Display all installed and no core modules

  ```
  drupal mod --status=installed --type=no-core
  ```

debug:multisite

List all multisites available in system

Usage:

```
drupal debug:multisite
dmu
```

Examples

- Displays multisite information

  ```
  drupal debug:multisite
  ```

debug:permission

Displays all user permissions.

Usage:

```
drupal debug:permission [arguments]
dp
```

Available arguments

Argument	Details
role	User role

Examples

- Displays all the permissions availables on the site

  ```
  drupal debug:permission
  ```

debug:plugin

Displays all plugin types.

Usage:

```
drupal debug:plugin [arguments]
dpl
```

Available arguments

Argument	Details
type	Plugin type
id	Plugin ID

Examples

- Displays a list with all the plugins on the current site

  ```
  drupal debug:plugin
  ```

- Displays block plugin information

  ```
  drupal debug:plugin block
  ```

- Displays block broken information

  ```
  drupal debug:plugin block broken
  ```

debug:queue

Displays the queues of your application

Usage:

```
drupal debug:queue
dq
```

Examples

- Displays the queues of the application

  ```
  drupal debug:queue
  ```

debug:rest

Display current rest resource for the application

Usage:

```
drupal debug:rest [arguments] [options]
rede
```

Available options

Option	Details	
--authorization	Rest resource status enabled	disabled

Available arguments

Argument	Details
resource-id	Rest ID

Examples

- Displays rest hooks

  ```
  drupal debug:rest
  ```

debug:router

Displays current routes for the application or information for a particular route

Usage:

```
drupal debug:router [arguments]
dr
```

Available arguments

Argument	Details
route-name	Route names

Examples

- Displays current routes for the application

  ```
  drupal rod
  ```

- Displays details for the route user.page (/user)

  ```
  drupal rod user.page
  ```

debug:settings

List user Drupal Console settings.

Usage:

```
drupal debug:settings
dse
```

Examples

```
drupal debug:settings
```

debug:site

List all known local and remote sites.

Usage:

```
drupal debug:site [arguments]
dsi
```

Available arguments

Argument	Details
target	Target
property	Property

Examples

```
drupal debug:site
```

debug:state

Show the current State keys.

Usage:

```
drupal debug:state [arguments]
dst
```

Available arguments

Argument	Details
key	The State key to debug.

Examples

- List of the states on the site

  ```
  drupal debug:state
  ```

- Displays a detail of the state install_task tok from the list of states

  ```
  drupal debug:state install_task
  ```

debug:test

List Test Units available for the application.

Usage:

```
drupal debug:test [arguments] [options]
td
```

Available options

Option	Details
--test-class	Test Class

Available arguments

Argument	Details
group	Group

Examples

```
drupal debug:test
```

debug:theme

Displays current themes for the application

Usage:

```
drupal debug:theme [arguments]
dt
```

Available arguments

Argument	Details
theme	Specific theme to debug

Examples

- List of themes on the site

  ```
  drupal debug:theme
  ```

- Bartik theme information

  ```
  drupal debug:theme bartik
  ```

debug:update

Displays current updates available for the application

Usage:

```
drupal debug:update
du
```

Examples

- List of pending updates

  ```
  drupal debug:update
  ```

debug:user

Displays current users for the application

Usage:

```
drupal debug:user [options]
dus
```

Available options

Option	Details
--uid	Filters the result list by uids [between quotes separated by spaces]
--username	Filters the result list by usernames [between quotes separated by spaces]
--mail	Filters the result list by user's e-mail [between quotes separated by spaces]
--roles	Roles to filter debug
--limit	How many users would you like to be listed in debug

Examples

- Users list on the site

  ```
  drupal debug:user
  ```

debug:views

Displays current views resources for the application

Usage:

```
drupal debug:views [arguments] [options]
vde
```

Available options

Option	Details	
--tag	View tag	
--status	View status (Enabled	Disabled)

Available arguments

Argument	Details
view-id	View ID

Examples

- List of views on the site

  ```
  drupal debug:views
  ```

debug:views:plugins

Displays current views plugins for the application

Usage:

```
drupal debug:views:plugins [arguments]
dvp
```

Available arguments

Argument	Details
type	Filter views plugins by type

Examples

- List of views plugins

  ```
  drupal debug:views:plugins
  ```

devel:dumper

Change the devel dumper plugin

Usage:

```
drupal devel:dumper [arguments]
dd
```

Available arguments

Argument	Details
dumper	Name of the devel dumper plugin

develop:contribute

Usage:

```
drupal develop:contribute [options]
```

Available options

Option	Details
--code-directory	commands.develop.contribute.options.code-directory

develop:example

Usage:

```
drupal develop:example
```

develop:example:container:aware

Usage:

```
drupal develop:example:container:aware
```

develop:gitbook

Update gitbook

Usage:

```
drupal develop:gitbook [arguments] [options]
```

Available options

Option	Details
--directory	directory
--help	Display this help message
--quiet	Do not output any message
--verbose	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
--version	Display this application version
--ansi	Force ANSI output
--no-ansi	Disable ANSI output
--no-interaction	Do not ask any interactive question
--env	The Environment name
--root	Define the Drupal root to be used in command execution
--debug	application.options.debug
--learning	Generate a verbose code output
--generate-chain	Shows command options and arguments as yaml output to be used in chain command
--generate-inline	Shows command options and arguments as inline command
--generate-doc	Shows command options and arguments as markdown
--target	Site name you want to interact with (for local or remote sites)
--uri	URI of the Drupal site to use (for multi-site environments or when running on an alternate port)
--yes	Skip confirmation and proceed

Available arguments

Argument	Details
command	The command to execute

dotenv:debug

Debug Dotenv debug values.

Usage:

```
drupal dotenv:debug
```

dotenv:init

Dotenv initializer.

Usage:

```
drupal dotenv:init
```

entity:delete

Delete an specific entity

Usage:

```
drupal entity:delete [arguments]
ed
```

Available arguments

Argument	Details
entity-definition-id	Entity definition id
entity-id	Entity ID to be deleted

Examples

- Delete entity type content using node id

  ```
  drupal entity:delete node 1
  ```

extend:example:one

Drupal Console extend example

Usage:

```
drupal extend:example:one
```

extend:example:two

Drupal Console extend example

Usage:

```
drupal extend:example:two
```

features:import

Import module config.

Usage:

```
drupal features:import [arguments] [options]
fei
```

Available options

Option	Details
--bundle	Bundle name

Available arguments

Argument	Details
packages	Package name

field:info

View information about fields.

Usage:

```
drupal field:info [options]
fii
```

Available options

Option	Details
--detailed	Extended output with machine names and descriptions
--entity	Restrict to a specific fieldabe entity type, for example: node, comment, taxonomy_term, shortcut, block_content, contact_message
--bundle	Restrict to a specific bundle type, for example: article

generate:authentication:provider

Generate an Authentication Provider

Usage:

```
drupal generate:authentication:provider [options]
gap
```

Available options

Option	Details
--module	The Module name.
--class	Authentication Provider class
--provider-id	Provider ID

Examples

- Generate an authentication provider specifying the module, the class and the provider id

```
drupal generate:authentication:provider  \
--module="modulename"  \
--class="DefaultAuthenticationProvider"  \
--provider-id="default_authentication_provider"
```

generate:breakpoint

Generate breakpoint

Usage:

```
drupal generate:breakpoint [options]
gb
```

Available options

Option	Details
--theme	Theme name
--breakpoints	Breakpoints

Examples

- Generate a breakpoint specifying the theme, a breakpoint name, its label, the media query, its weight and multipliers

```
drupal generate:breakpoint  \
--theme="classy"  \
--breakpoints='"breakpoint_name":"narrow", "breakpoint_label":"narrow", "breakpoint_m
```

generate:cache:context

Generate a cache context

Usage:

```
drupal generate:cache:context [options]
gcc
```

Available options

Option	Details
--module	The Module name.
--cache-context	Enter the cache context name
--class	Cache context class name
--services	Load services from the container.

Examples

- Generate cache for a context specifying the module, the context name and its class

```
drupal generate:cache:context  \
--module="modulename"  \
--cache-context="ContextName"  \
--class="DefaultCacheContext"
```

generate:command

Generate commands for the console.

Usage:

```
drupal generate:command [options]
gco
gcm
```

Available options

Option	Details
--extension	The extension name.
--extension-type	The extension type.
--class	The Class that describes the command. (Must end with the word 'Command').
--name	The Command name.
--container-aware	Is the command aware of the drupal site installation when executed
--services	Load services from the container.

Examples

- Generate a command specifying the extension name and type, its class and the name.

```
drupal generate:command  \
--extension="ExtensionName"  \
--extension-type="module"  \
--class="DefaultCommand"  \
--name="CommandName"
```

generate:controller

Generate & Register a controller

Usage:

```
drupal generate:controller [options]
gcon
gcn
```

Available options

Option	Details
--module	The Module name.
--class	Controller Class name
--routes	The routes, must be an array containing [title, method, path]
--services	Load services from the container.
--test	Generate a test class

Examples

- Generate controller specifying the module name, the class name and its routes

```
drupal generate:controller  \
--module="modulename"  \
--class="DefaultController"  \
--routes='"title":"ControllerMethod", "name":"modulename.default_controller_hello", '
--test
```

generate:doc:cheatsheet

commands.generate.doc.cheatsheet.description

Usage:

```
drupal generate:doc:cheatsheet [options]
gdc
```

Available options

Option	Details
--path	commands.generate.doc.cheatsheet.options.path
--wkhtmltopdf	commands.generate.doc.cheatsheet.options.wkhtmltopdf

generate:doc:dash

commands.generate.doc.dash.description

Usage:

```
drupal generate:doc:dash [options]
gdd
```

Available options

Option	Details
--path	commands.generate.doc.dash.options.path

generate:doc:data

commands.generate.doc.data.description

Usage:

```
drupal generate:doc:data [options]
gdda
```

Available options

Option	Details
--file	commands.generate.doc.data.options.file

generate:doc:gitbook

commands.generate.doc.gitbook.description

Usage:

```
drupal generate:doc:gitbook [arguments] [options]
gdg
```

Available options

Option	Details
--path	commands.generate.doc.gitbook.options.path
--help	Display this help message
--quiet	Do not output any message
--verbose	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
--version	Display this application version
--ansi	Force ANSI output
--no-ansi	Disable ANSI output
--no-interaction	Do not ask any interactive question
--env	The Environment name
--root	Define the Drupal root to be used in command execution
--debug	application.options.debug
--learning	Generate a verbose code output
--generate-chain	Shows command options and arguments as yaml output to be used in chain command
--generate-inline	Shows command options and arguments as inline command
--generate-doc	Shows command options and arguments as markdown
--target	Site name you want to interact with (for local or remote sites)
--uri	URI of the Drupal site to use (for multi-site environments or when running on an alternate port)
--yes	Skip confirmation and proceed

Available arguments

Argument	Details
command	The command to execute

generate:entity:bundle

Generate a new content type (node / entity bundle)

Usage:

```
drupal generate:entity:bundle [options]
geb
```

Available options

Option	Details
--module	The Module name.
--bundle-name	The content type's machine name
--bundle-title	The content type's human-readable name

Examples

- Generate bundle entity specifying the module, the bundle name and its title

```
drupal generate:entity:bundle  \
--module="modulename"  \
--bundle-name="default"  \
--bundle-title="default"
```

generate:entity:config

Generate a new config entity

Usage:

```
drupal generate:entity:config [options]
gec
gecg
```

Available options

Option	Details
--module	The Module name.
--entity-class	The config entity class
--entity-name	The config entity name
--base-path	The base-path for the config entity routes
--label	The label
--bundle-of	Acts as bundle for content entities

Examples

- Generate config entity specifying the module, the entity class, the entity name, its path and label

```
drupal generate:entity:config  \
--module="modulename"  \
--entity-class="DefaultEntity"  \
--entity-name="default_entity"  \
--base-path="/admin/structure"  \
--label="Default entity"
```

generate:entity:content

Generate a new content entity

Usage:

```
drupal generate:entity:content [options]
geco
gect
```

Available options

Option	Details
--module	The Module name.
--entity-class	The content entity class
--entity-name	The content entity name
--base-path	The base-path for the content entity routes
--label	The label
--has-bundles	Entity has bundles
--is-translatable	Content entity translatable
--revisionable	commands.generate.entity.content.options.revisionable

Examples

- Generate a content entity specifying the module, the entity class, the entity name, its path and label

  ```
  drupal generate:entity:content  \
  --module="modulename"  \
  --entity-class="DefaultEntity"  \
  --entity-name="default_entity"  \
  --base-path="/admin/structure"  \
  --label="Default entity"
  ```

- Generate a translatable and revisionable content entity specifying the module, the entity class, the entity name, its path and label

```
drupal generate:entity:content  \
--module="modulename"  \
--entity-class="DefaultEntity"  \
--entity-name="default_entity"  \
--base-path="/admin/structure"  \
--label="Default entity"  \
--is-translatable  \
--revisionable
```

generate:event:subscriber

Generate an event subscriber

Usage:

```
drupal generate:event:subscriber [options]
ges
```

Available options

Option	Details
--module	The Module name.
--name	commands.generate.service.options.name
--class	Class name
--events	Load events from the container
--services	Load services from the container.

Examples

- Generate an event subscriber specifying the module name, its name, the class and the events to subscribe

```
drupal generate:event:subscriber  \
--module="modulename"  \
--name="modulename.default"  \
--class="DefaultSubscriber"  \
--events='kernel_request'
```

generate:form

Generate a new "%s"

Usage:

```
drupal generate:form [options]
gf
```

Available options

Option	Details
--module	The Module name.
--class	The form class name
--form-id	The Form id
--services	Load services from the container.
--config-file	Add a config file
--inputs	Create inputs in a form.
--path	Enter the form path
--menu-link-gen	Generate a menu link
--menu-link-title	A title for the menu link
--menu-parent	Menu parent
--menu-link-desc	A description for the menu link

Examples

- Generate an empty form with config file specifying the module name, the class, a form id and the path

```
drupal generate:form  \
--module="modulename"  \
--class="DefaultForm"  \
--form-id="default_form"  \
--config-file  \
--path="/modulename/form/default"
```

- Generate a form with 2 fields and a config file specifying the module name, the class, a form id, the inputs and the path

```
drupal generate:form  \
--module="modulename"  \
--class="DefaultForm"  \
--form-id="default_form"  \
--config-file  \
--inputs='"name":"inputname", "type":"text_format", "label":"InputName", "options":"'
--inputs='"name":"email", "type":"email", "label":"Email", "options":"", "description
--path="/modulename/form/default"
```

generate:form:alter

Generate an implementation of hook_form_alter() or hook_form_FORM_ID_alter

Usage:

```
drupal generate:form:alter [options]
gfa
```

Available options

Option	Details
--module	The Module name.
--form-id	Form ID to alter
--inputs	Create inputs in a form.

Examples

- Generate a hook form alter for an empty form specifying the module name

```
drupal generate:form:alter  \
--module="modulename"
```

- Generate a hook form alter with 2 fields specifying the module name and the inputs

```
drupal generate:form:alter  \
--module="modulename"  \
--inputs='"name":"inputtext", "type":"text_format", "label":"InputText", "options":"'
--inputs='"name":"email", "type":"email", "label":"Email", "options":"", "descriptior
```

generate:form:config

commands.generate.form.description

Usage:

```
drupal generate:form:config [options]
gf
gfc
```

Available options

Option	Details
--module	The Module name.
--class	The form class name
--form-id	The Form id
--services	Load services from the container.
--config-file	Add a config file
--inputs	Create inputs in a form.
--path	Enter the form path
--menu-link-gen	Generate a menu link
--menu-link-title	A title for the menu link
--menu-parent	Menu parent
--menu-link-desc	A description for the menu link

generate:help

Generate an implementation of hook_help()

Usage:

```
drupal generate:help [options]
gh
```

Available options

Option	Details
--module	The Module name.
--description	Module description

Examples

- Generate a hook help specifying the module name and the description

  ```
  drupal generate:help  \
  --module="modulename"  \
  --description="My Awesome Module"
  ```

generate:module

Generate a module.

Usage:

```
drupal generate:module [options]
gm
```

Available options

Option	Details
--module	The Module name
--machine-name	The machine name (lowercase and underscore only)
--module-path	The path of the module
--description	Module description
--core	Core version
--package	Module package
--module-file	Add a .module file
--features-bundle	Define module as feature using the given Features bundle name
--composer	Add a composer.json file
--dependencies	Module dependencies separated by commas (i.e. context, panels)
--test	Generate a test class
--twigtemplate	Generate theme template

Examples

- Generate a module specifying the module name, machine name, the path, its description, drupal core and the package name. In this example the composer file, the unit test and twig template are generated too

```
drupal generate:module  \
--module="modulename"  \
--machine-name="modulename"  \
--module-path="/modules/custom"  \
--description="My Awesome Module"  \
--core="8.x"  \
--package="Custom"  \
--module-file  \
--composer  \
--test  \
--twigtemplate
```

generate:module:file

Generate a .module file

Usage:

```
drupal generate:module:file [options]
gmf
```

Available options

Option	Details
--module	The Module name.

Examples

- Generate the .module file specifying the module name

  ```
  drupal generate:module:file  \
  --module="modulename"
  ```

generate:permissions

commands.generate.permission.description

Usage:

```
drupal generate:permissions [options]
gp
```

Available options

Option	Details
--module	The Module name.
--permissions	Create permissions.

generate:plugin:block

Generate a plugin block

Usage:

```
drupal generate:plugin:block [options]
gpb
```

Available options

Option	Details
--module	The Module name.
--class	Plugin class name
--label	Plugin label
--plugin-id	Plugin id
--theme-region	Theme region to render Plugin Block
--inputs	Create inputs in a form.
--services	Load services from the container.

Examples

- Generate a plugin block in the header region with an input field specifying the module name, the class, the label, its id, the region and the input

```
drupal generate:plugin:block  \
--module="modulename"  \
--class="DefaultBlock"  \
--label="Default block"  \
--plugin-id="default_block"  \
--theme-region="header"  \
--inputs='"name":"inputtext", "type":"text_format", "label":"InputText", "options":"'
```

generate:plugin:ckeditorbutton

Generate CKEditor button plugin.

Usage:

```
drupal generate:plugin:ckeditorbutton [options]
gpc
```

Available options

Option	Details
--module	The Module name.
--class	Plugin class name
--label	Plugin label
--plugin-id	Plugin ID. NOTE: This corresponds to the CKEditor plugin name. It is the first argument of the CKEDITOR.plugins.add() function in the plugin.js file.
--button-name	Button name. NOTE: This corresponds to the CKEditor button name. They are the first argument of the editor.ui.addButton() or editor.ui.addRichCombo() functions in the plugin.js file.
--button-icon-path	Button icon path. This is the path to the icon/image of the button.

Examples

- Generate CKEditor button specifying the module name, the class, the label, its id, the button name and the icon path

```
drupal generate:plugin:ckeditorbutton  \
--module="modulename"  \
--class="DefaultCKEditorButton"  \
--label="Default ckeditor button"  \
--plugin-id="default ckeditor button"  \
--button-name="Default ckeditor button"  \
--button-icon-path="modules/custom/modulename/js/plugins/default ckeditor button/imag
```

generate:plugin:condition

Generate a plugin condition.

Usage:

```
drupal generate:plugin:condition [options]
gpco
gpc
```

Available options

Option	Details
--module	The Module name.
--class	Plugin condition class name
--label	Plugin condition label
--plugin-id	Plugin condition id
--context-definition-id	Context definition ID
--context-definition-label	Context definition label
--context-definition-required	Context definition is required (TRUE/FALSE)

Examples

- Generate a plugin condition for a node entity type specifying the module name, the class, the label, its id and the context definition

  ```
  drupal generate:plugin:condition  \
  --module="modulename"  \
  --class="ExampleCondition"  \
  --label="Example condition"  \
  --plugin-id="example_condition"  \
  --context-definition-id="entity:node"  \
  --context-definition-label="node"  \
  --context-definition-required
  ```

- Generate a plugin condition for language specifying the module name, the class, the label, its id and the context definition

```
drupal generate:plugin:condition  \
--module="modulename"  \
--class="ExampleCondition"  \
--label="Example condition"  \
--plugin-id="example_condition"  \
--context-definition-id="language"  \
--context-definition-label="Language"  \
--context-definition-required
```

- Generate a plugin condition for role configuration specifying the module name, the class, the label, its id and the context definition

```
drupal generate:plugin:condition  \
--module="modulename"  \
--class="ExampleCondition"  \
--label="Example condition"  \
--plugin-id="example_condition"  \
--context-definition-id="entity:user_role"  \
--context-definition-label="user_role"  \
--context-definition-required
```

generate:plugin:field

Generate field type, widget and formatter plugins.

Usage:

```
drupal generate:plugin:field [options]
gpf
```

Available options

Option	Details
--module	The Module name.
--type-class	Field type plugin class name
--type-label	Field type plugin label
--type-plugin-id	Field type plugin id
--type-description	commands.generate.plugin.field.options.type-type-description
--formatter-class	commands.generate.plugin.field.options.class
--formatter-label	Field formatter plugin label
--formatter-plugin-id	Field formatter plugin id
--widget-class	Field formatter plugin class name
--widget-label	Field widget plugin label
--widget-plugin-id	Field widget plugin id
--field-type	Field type the formatter and widget plugin can be used with
--default-widget	Default field widget of the field type plugin
--default-formatter	Default field formatter of field type plugin

Examples

- Generate field type, widget and formatter plugins specifying the module name, the type (class, label, plugin id and description), the formatter (class, label, plugin id) and the widget (class, label and plugin id)

```
drupal generate:plugin:field  \
--module="modulename"  \
--type-class="ExampleFieldType"  \
--type-label="Example field type"  \
--type-plugin-id="example_field_type"  \
--type-description="My Field Type"  \
--formatter-class="ExampleFormatterType"  \
--formatter-label="Example formatter type"  \
--formatter-plugin-id="example_formatter_type"  \
--widget-class="ExampleWidgetType"  \
--widget-label="Example widget type"  \
--widget-plugin-id="example_widget_type"  \
--field-type="example_field_type"  \
--default-widget="example_widget_type"  \
--default-formatter="example_formatter_type"
```

generate:plugin:fieldformatter

Generate field formatter plugin.

Usage:

```
drupal generate:plugin:fieldformatter [options]
gpff
```

Available options

Option	Details
--module	The Module name.
--class	Plugin class name
--label	Plugin label
--plugin-id	Plugin id
--field-type	Field type the plugin can be used with

Examples

- Generate a a text field formatter plugin specifying the module name, the class, the label its plugin id and the field type

```
drupal generate:plugin:fieldformatter  \
--module="modulename"  \
--class="ExampleFieldFormatter"  \
--label="Example field formatter"  \
--plugin-id="example_field_formatter"  \
--field-type="text"
```

generate:plugin:fieldtype

Generate field type plugin.

Usage:

```
drupal generate:plugin:fieldtype [options]
gpft
```

Available options

Option	Details
--module	The Module name.
--class	Plugin class name
--label	Plugin label
--plugin-id	Plugin id
--description	Plugin Description
--default-widget	Default field widget of this plugin
--default-formatter	Default field formatter of this plugin

Examples

- Generate a field type plugin specifying the module name, the class, its label, the plugin id and a description

  ```
  drupal generate:plugin:fieldtype  \
  --module="modulename"  \
  --class="ExampleFieldType"  \
  --label="Example field type"  \
  --plugin-id="example_field_type"  \
  --description="My Field Type"
  ```

- Generate a field type plugin with a default widget and formatter specifying the module name, the class, its label, the plugin id and a description

```
drupal generate:plugin:fieldtype  \
--module="modulename"  \
--class="ExampleFieldType"  \
--label="Example field type"  \
--plugin-id="example_field_type"  \
--description="My Field Type"  \
--default-widget="DefaultWidget"  \
--default-formatter="DefaultFormatter"
```

generate:plugin:fieldwidget

Generate field widget plugin.

Usage:

```
drupal generate:plugin:fieldwidget [options]
gpfw
```

Available options

Option	Details
--module	The Module name.
--class	Plugin class name
--label	Plugin label
--plugin-id	Plugin id
--field-type	Field type the plugin can be used with

Examples

- Generate a text type field widget plugin specifying the module name, the class, its label, the plugin id and the field type

```
drupal generate:plugin:fieldwidget  \
--module="modulename"  \
--class="ExampleFieldWidget"  \
--label="Example field widget"  \
--plugin-id="example_field_widget"  \
--field-type="text"
```

generate:plugin:imageeffect

Generate image effect plugin.

Usage:

```
drupal generate:plugin:imageeffect [options]
gpie
```

Available options

Option	Details
--module	The Module name.
--class	Plugin class name
--label	Plugin label
--plugin-id	Plugin id
--description	Plugin Description

Examples

- Generate a image effect plugin specifying the module name, the class, its label, the plugin id and a description

```
drupal generate:plugin:imageeffect  \
--module="modulename"  \
--class="DefaultImageEffect"  \
--label="Default image effect"  \
--plugin-id="default_image_effect"  \
--description="My Image Effect"
```

generate:plugin:imageformatter

Generate image formatter plugin.

Usage:

```
drupal generate:plugin:imageformatter [options]
gpif
```

Available options

Option	Details
--module	The Module name.
--class	Plugin class name
--label	Plugin label
--plugin-id	Plugin id

Examples

- Generate a image formatter plugin specifying the module name, the class, its label and the plugin id

```
drupal generate:plugin:imageformatter  \
--module="modulename"  \
--class="ExampleImageFormatter"  \
--label="Example image formatter"  \
--plugin-id="example_image_formatter"
```

generate:plugin:mail

Generate a plugin mail

Usage:

```
drupal generate:plugin:mail [options]
gpm
```

Available options

Option	Details
--module	The Module name.
--class	Plugin class name
--label	Plugin label
--plugin-id	Plugin id
--services	Load services from the container.

Examples

- Generate an email plugin specifying the module name, the class, its label and the plugin id

```
drupal generate:plugin:mail  \
--module="modulename"  \
--class="HtmlFormatterMail"  \
--label="Html formatter mail"  \
--plugin-id="html_formatter_mail"
```

generate:plugin:migrate:process

Generate a migrate process plugin

Usage:

```
drupal generate:plugin:migrate:process [options]
gpmp
```

Available options

Option	Details
--module	The Module name.
--class	Plugin class name
--plugin-id	Plugin id

Examples

- Generate a migration plugin process specifying the module name, the class and its id

```
drupal generate:plugin:migrate:process  \
--module="modulename"  \
--class="MigrationProcess"  \
--plugin-id="migrationprocess"
```

generate:plugin:migrate:source

Generate a migrate source plugin

Usage:

```
drupal generate:plugin:migrate:source [options]
gpms
```

Available options

Option	Details
--module	The Module name.
--class	Plugin class name
--plugin-id	Plugin id
--table	Table to query
--alias	Short alias to refer to the table as
--group-by	Field to group results by
--fields	Fields to export

Examples

- Generate a migration source plugin specifying the module name, the class, its plugin id, the table and its alias

  ```
  drupal generate:plugin:migrate:source  \
  --module="modulename"  \
  --class="PluginClassName"  \
  --plugin-id="plugin_class_name"  \
  --table="DefaultTableName"  \
  --alias="D"
  ```

- Generate a migration source plugin for specific fields of the users table specifying the module name, the class, its plugin id, the table, its alias and the fields

```
drupal generate:plugin:migrate:source  \
--module="modulename"  \
--class="DefaultPluginClass"  \
--plugin-id="default_plugin_class"  \
--table="users"  \
--alias="u"  \
--fields='"id":"id", "description":"the user id"'  \
--fields='"id":"username", "description":"the username"'  \
--fields='"id":"password", "description":"the user password"'  \
--fields='"id":"email", "description":"the user email"'
```

generate:plugin:rest:resource

Generate plugin rest resource

Usage:

```
drupal generate:plugin:rest:resource [options]
gprr
```

Available options

Option	Details
--module	The Module name.
--class	Plugin Rest Resource class
--name	commands.generate.service.options.name
--plugin-id	Plugin Rest Resource id
--plugin-label	Plugin Rest Resource Label
--plugin-url	Plugin Rest Resource URL
--plugin-states	Plugin Rest Resource States

Examples

- Generate a rest resource plugin using GET specifying the module name, the class, the plugin id, its label, the target url and the request type

```
drupal generate:plugin:rest:resource  \
--module="modulename"  \
--class="DefaultRestResource"  \
--plugin-id="default_rest_resource"  \
--plugin-label="Default rest resource"  \
--plugin-url="http://rest.resources.example.com"  \
--plugin-states='GET'
```

generate:plugin:rulesaction

Generate a plugin rule action

Usage:

```
drupal generate:plugin:rulesaction [options]
gpra
```

Available options

Option	Details
--module	The Module name.
--class	Plugin class name
--label	Plugin label
--plugin-id	Plugin id
--type	Action Type (user or node)
--category	Plugin category
--context	Plugin context

Examples

- Generate a user rule action plugin specifying the module name, the class, its label, the plugin id, the type, the category and its context

  ```
  drupal generate:plugin:rulesaction  \
  --module="modulename"  \
  --class="DefaultAction"  \
  --label="Default action"  \
  --plugin-id="default_action"  \
  --type="user"  \
  --category="default_action"  \
  --context="default_action"
  ```

- Generate a node rule action plugin specifying the module name, the class, its label, the plugin id, the type, the category and its context

```
drupal generate:plugin:rulesaction  \
--module="modulename"  \
--class="DefaultAction"  \
--label="Default action"  \
--plugin-id="default_action"  \
--type="node"  \
--category="default_action" \
--context="default_action"
```

generate:plugin:skeleton

Generate an implementation of a skeleton plugin

Usage:

```
drupal generate:plugin:skeleton [options]
gps
```

Available options

Option	Details
--module	The Module name.
--plugin-id	commands.generate.plugin.options.plugin-id
--class	Plugin class name
--services	Load services from the container.

Examples

- Generate a plugin skeleton specifying module name, the plugin id and the class

```
drupal generate:plugin:skeleton  \
--module="modulename"  \
--plugin-id="link_relation_type"  \
--class="DefaultLinkRelationType"
```

generate:plugin:type:annotation

Generate a plugin type with annotation discovery

Usage:

```
drupal generate:plugin:type:annotation [options]
gpta
```

Available options

Option	Details
--module	The Module name.
--class	Plugin type class name
--machine-name	commands.generate.plugin.type.annotation.options.plugin-id
--label	Plugin type label

Examples

- Generate a plugin with annotation discovery specifying module name, class name, machine name and label

```
drupal generate:plugin:type:annotation  \
--module="modulename"  \
--class="ExamplePlugin"  \
--machine-name="example_plugin"  \
--label="Example plugin"
```

generate:plugin:type:yaml

Generate a plugin type with Yaml discovery

Usage:

```
drupal generate:plugin:type:yaml [options]
gpty
```

Available options

Option	Details
--module	The Module name.
--class	Plugin type class name
--plugin-name	Plugin type machine name
--plugin-file-name	Plugin file name

Examples

- Generate a plugin with Yaml discovery specifying module name, class name, plugin name and plugin file name

```
drupal generate:plugin:type:yaml  \
--module="modulename"  \
--class="ExamplePlugin"  \
--plugin-name="example_plugin"  \
--plugin-file-name="example.plugin"
```

generate:plugin:views:field

Generate a custom plugin view field.

Usage:

```
drupal generate:plugin:views:field [options]
gpvf
```

Available options

Option	Details
--module	The Module name.
--class	Views plugin field class name
--title	Views plugin field title
--description	Views plugin field description

Examples

- Generate a custom view field plugin specifying the module name, the class, a title and its description

```
drupal generate:plugin:views:field  \
--module="modulename"  \
--class="CustomViewsField"  \
--title="Custom views field"  \
--description="My awesome custom views field plugin."
```

generate:post:update

commands.generate.post:update.description

Usage:

```
drupal generate:post:update [options]
gpu
```

Available options

Option	Details
--module	The Module name.
--post-update-name	Post Update Name

Examples

- Generate an implementation of post update hook specifying the module name and the post update name

  ```
  drupal generate:post:update  \
  --module="modulename"  \
  --post-update-name="PostUpdateName"
  ```

generate:profile

Generate a profile.

Usage:

```
drupal generate:profile [options]
gpr
```

Available options

Option	Details
--profile	The profile name
--machine-name	The machine name (lowercase and underscore only)
--description	Profile description
--core	Core version
--dependencies	Module dependencies separated by commas (i.e. context, panels)
--themes	commands.generate.profile.options.themes
--distribution	The distribution name

Examples

- Generate a profile specifying the profile name, the machine name, a description, the core and its module dependencies

```
drupal generate:profile  \
--profile="NewProfileName"  \
--machine-name="newprofilename"  \
--description="My Useful Profile"  \
--core="8.x"  \
--dependencies="modulename"
```

generate:routesubscriber

Generate a RouteSubscriber

Usage:

```
drupal generate:routesubscriber [options]
gr
```

Available options

Option	Details
--module	The Module name.
--name	Service name
--class	Class name

Examples

- Generate a route subscriber specifying the module name, the route name and its class

```
drupal generate:routesubscriber  \
--module="modulename"  \
--name="modulename.route_subscriber"  \
--class="RouteSubscriber"
```

generate:service

Generate service

Usage:

```
drupal generate:service [options]
gs
```

Available options

Option	Details
--module	The Module name.
--name	commands.generate.service.options.name
--class	Class name
--interface	commands.common.service.options.interface
--interface-name	commands.common.service.options.interface-name
--services	Load services from the container.
--path-service	Path

Examples

- Generate a services without interface specifying the module name, the service name, the class and its path

  ```
  drupal generate:service  \
  --module="modulename"  \
  --name="modulename.default"  \
  --class="DefaultService"  \
  --path-service="/modules/custom/modulename/src/"
  ```

- Generate a services with interface specifying the module name, the service name, the class, the interface name and its path

```
drupal generate:service  \
--module="modulename"  \
--name="modulename.default"  \
--class="DefaultService"  \
--interface  \
--interface-name="InterfaceName"  \
--path-service="/modules/custom/modulename/src/"
```

generate:theme

Generate a theme.

Usage:

```
drupal generate:theme [options]
gt
```

Available options

Option	Details
--theme	commands.generate.theme.options.module
--machine-name	The machine name (lowercase and underscore only)
--theme-path	commands.generate.theme.options.module-path
--description	Theme description
--core	Core version
--package	Theme package
--global-library	Global styling library name
--libraries	commands.generate.theme.options.libraries
--base-theme	Base theme (i.e. classy, stable)
--regions	Regions
--breakpoints	Breakpoints

Examples

- Generate a theme without region and without breakpoint specifying the theme name, its machine name, the theme path, a description, the drupal core, the package name and the global library

```
drupal generate:theme  \
--theme="AnotherTheme"  \
--machine-name="anothertheme"  \
--theme-path="/themes/custom"  \
--description="My Awesome theme"  \
--core="8.x"  \
--package="PackageName"  \
--global-library="global-styling"  \
--base-theme="false"
```

- Generate a theme base on stable theme with two region defined and one breakpoint specifying the theme name, its machine name, the theme path, a description, the drupal core, the package name, a global library, its base, the regions and the breakpoint

```
drupal generate:theme  \
--theme="MyTheme"  \
--machine-name="mytheme"  \
--theme-path="/themes/custom"  \
--description="My Awesome theme"  \
--core="8.x"  \
--package="MyThemePackage"  \
--global-library="global-styling"  \
--base-theme="stable"  \
--regions='"region_name":"Content", "region_machine_name":"content"'  \
--regions='"region_name":"Panel", "region_machine_name":"panel"'  \
--breakpoints='"breakpoint_name":"narrow", "breakpoint_label":"narrow", "breakpoint_m
```

generate:twig:extension

Generate a Twig extension.

Usage:

```
drupal generate:twig:extension [options]
gte
```

Available options

Option	Details
--module	The Module name.
--name	Twig Extension name
--class	Class name
--services	Load services from the container.

Examples

- Generate a twig extension specifying the module name, the extension name and its class

  ```
  drupal generate:twig:extension  \
  --module="modulename"  \
  --name="modulename.twig.extension"  \
  --class="DefaultTwigExtension"
  ```

generate:update

Generate an implementation of hook_update_N()

Usage:

```
drupal generate:update [options]
gu
```

Available options

Option	Details
--module	The Module name.
--update-n	Update Number

Examples

- Generate an update N hook implementation specifying the module name and the N value

  ```
  drupal generate:update  \
  --module="modulename"  \
  --update-n="8001"
  ```

image:styles:flush

Execute flush function by image style or execute all flush images styles

Usage:

```
drupal image:styles:flush [arguments]
isf
```

Available arguments

Argument	Details
styles	The Images Styles name.

Examples

- Flush large image style

  ```
  drupal image:styles:flush  large
  ```

- Flush thumbnail image style

  ```
  drupal image:styles:flush  thumbnail
  ```

locale:language:add

Add a language to be supported by your site

Usage:

```
drupal locale:language:add [arguments]
```

Available arguments

Argument	Details
language	Language for instance es or Spanish

locale:language:delete

Delete a language to be supported by your site

Usage:

```
drupal locale:language:delete [arguments]
```

Available arguments

Argument	Details
language	Language for instance es or Spanish

locale:translation:status

List available translation updates

Usage:

```
drupal locale:translation:status [arguments]
```

Available arguments

Argument	Details
language	Language for instance es or Spanish

migrate:execute

Execute a migration available for application

Usage:

```
drupal migrate:execute [arguments] [options]
mie
```

Available options

Option	Details
--site-url	Site Source URL
--db-type	commands.migrate.setup.migrations.options.db-type
--db-host	Database Host
--db-name	Database Name
--db-user	Database User
--db-pass	Database Pass
--db-prefix	Database Prefix
--db-port	Database Port
--exclude	Migration id(s) to exclude
--source-base_path	Local file directory containing your source site (e.g. /var/www/docroot), or your site address (for example http://example.com)

Available arguments

Argument	Details
migration-ids	Migration id(s)

migrate:rollback

Rollback one or multiple migrations

Usage:

```
drupal migrate:rollback [arguments] [options]
mir
```

Available options

Option	Details
--source-base_path	commands.migrate.setup.options.source-base-path

Available arguments

Argument	Details
migration-ids	Migration id(s)

migrate:setup

Load and create the relevant migrations for a provided legacy database

Usage:

```
drupal migrate:setup [options]
mis
```

Available options

Option	Details
--db-type	Drupal Database type
--db-host	Database Host
--db-name	Database Name
--db-user	Database User
--db-pass	Database Pass
--db-prefix	Database Prefix
--db-port	Database Port
--source-base_path	commands.migrate.setup.options.source-base-path

module:dependency:install

commands.module.install.dependencies.description

Usage:

```
drupal module:dependency:install [arguments]
modi
```

Available arguments

Argument	Details
module	commands.module.install.dependencies.arguments.module

Examples

- Install the dependencies of the specfied module

  ```
  drupal module:dependency:install  modulename
  ```

module:download

Download module or modules in application

Usage:

```
drupal module:download [arguments] [options]
mod
md
```

Available options

Option	Details
--path	The path of the contrib project
--latest	Default to download most recent version
--composer	Download the module using Composer
--unstable	commands.module.install.options.unstable

Available arguments

Argument	Details
module	Module or modules to be enabled should be separated by a space

Examples

- Download module specifying module name and its path

  ```
  drupal module:download  modulename  \
  --path="modules/contrib"
  ```

module:install

Install module or modules in the application

Usage:

```
drupal module:install [arguments] [options]
moi
```

Available options

Option	Details
--latest	Default to download most recent version
--composer	Uninstalls the module using Composer
--help	Display this help message
--quiet	Do not output any message
--verbose	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
--version	Display this application version
--ansi	Force ANSI output
--no-ansi	Disable ANSI output
--no-interaction	Do not ask any interactive question
--env	The Environment name
--root	Define the Drupal root to be used in command execution
--debug	application.options.debug
--learning	Generate a verbose code output
--generate-chain	Shows command options and arguments as yaml output to be used in chain command
--generate-inline	Shows command options and arguments as inline command
--generate-doc	Shows command options and arguments as markdown
--target	Site name you want to interact with (for local or remote sites)
--uri	URI of the Drupal site to use (for multi-site environments or when running on an alternate port)
--yes	Skip confirmation and proceed

Available arguments

Argument	Details
command	The command to execute
module	Module or modules to be enabled should be separated by a space

Examples

- Install module specifying the module name

```
drupal module:install  modulename
```

module:path

Returns the relative path to the module (or absolute path)

Usage:

```
drupal module:path [arguments] [options]
mop
```

Available options

Option	Details
--absolute	Return module absolute path

Available arguments

Argument	Details
module	The Module name (machine name)

Examples

- Get the relative path of the module specifying the module name

  ```
  drupal module:path  modulename
  ```

module:uninstall

Uninstall module or modules in the application

Usage:

```
drupal module:uninstall [arguments] [options]
mou
```

Available options

Option	Details
--force	Do you want to ignore dependencies and forcefully uninstall the module?
--composer	Uninstalls the module using Composer

Available arguments

Argument	Details
module	Module name (press to stop adding modules)

Examples

- Uninstall the module specifying the module name

  ```
  drupal module:uninstall  modulename
  ```

module:update

Update core, module or modules in the application

Usage:

```
drupal module:update [arguments] [options]
moup
```

Available options

Option	Details
--composer	Update the module using Composer
--simulate	Simulate the update process with Composer

Available arguments

Argument	Details
module	Module or modules to be updated should be separated by a space. Leave empty for updating the core and all your modules managed by Composer.

Examples

- Update module specifying module name and composer parameter

```
drupal module:update  modulename  \
--composer
```

multisite:new

Sets up the files for a new multisite install.

Usage:

```
drupal multisite:new [arguments] [options]
mun
```

Available options

Option	Details
--copy-default	Copies existing site from the default install.

Available arguments

Argument	Details
directory	Name of directory under 'sites' which should be created.
uri	Site URI to add to sites.php.

Examples

- Set up files for a multisite install specifying destination path and uri

  ```
  drupal multisite:new  vendor/newsite http://mysite.example.com
  ```

node:access:rebuild

Rebuild node access permissions.

Usage:

```
drupal node:access:rebuild [options]
nar
```

Available options

Option	Details
--batch	Process in batch mode.

Examples

- Rebuild node access permissions

  ```
  drupal node:access:rebuild --batch
  ```

queue:run

Process the selected queue.

Usage:

```
drupal queue:run [arguments]
qr
```

Available arguments

Argument	Details
name	Queue name.

quick:start

Download, install and serve a new Drupal project

Usage:

```
drupal quick:start [options]
```

Available options

Option	Details
--repository	repository
--directory	directory
--profile	profile

rest:disable

Disable a rest resource for the application

Usage:

```
drupal rest:disable [arguments]
red
redi
```

Available arguments

Argument	Details
resource-id	Rest ID

rest:enable

Enable a rest resource for the application

Usage:

```
drupal rest:enable [arguments]
ree
```

Available arguments

Argument	Details
resource-id	Rest ID

router:rebuild

Rebuild routes for the application

Usage:

```
drupal router:rebuild
rr
ror
```

Examples

- Rebuild routes

  ```
  drupal router:rebuild
  ```

sample:default

commands.sample.default.description

Usage:

```
drupal sample:default
```

settings:set

Change a specific setting value in DrupalConsole config file

Usage:

```
drupal settings:set [arguments] [options]
```

Available options

Option	Details
--help	Display this help message
--quiet	Do not output any message
--verbose	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
--version	Display this application version
--ansi	Force ANSI output
--no-ansi	Disable ANSI output
--no-interaction	Do not ask any interactive question
--env	The Environment name
--root	Define the Drupal root to be used in command execution
--debug	application.options.debug
--learning	Generate a verbose code output
--generate-chain	Shows command options and arguments as yaml output to be used in chain command
--generate-inline	Shows command options and arguments as inline command
--generate-doc	Shows command options and arguments as markdown
--target	Site name you want to interact with (for local or remote sites)
--uri	URI of the Drupal site to use (for multi-site environments or when running on an alternate port)
--yes	Skip confirmation and proceed

Available arguments

Argument	Details
command	The command to execute
name	Setting name in YAML flatten format to set a value in Drupal Console config file
value	Setting value to set in Drupal Console config file

Examples

- Set application language setting value to "es"

```
drupal settings:set  application.language es
```

site:import:local

Import/Configure an existing local Drupal project

Usage:

```
drupal site:import:local [arguments] [options]
sil
```

Available options

Option	Details
--environment	Name of the environment that is going to be imported

Available arguments

Argument	Details
name	Name that will be used to generate the site config
directory	Existing Drupal root directory

Examples

- Import local drupal project specifying the site name and the path

  ```
  drupal site:import:local  SiteName /private/var/www/vhost/anexusit/drupal8.dev/web
  ```

site:install

Install a Drupal project

Usage:

```
drupal site:install [arguments] [options]
si
```

Available options

Option	Details
--langcode	commands.site.install.options.langcode
--db-type	commands.site.install.options.db-type
--db-file	commands.site.install.options.db-file
--db-host	Database Host
--db-name	Database Name
--db-user	Database User
--db-pass	Database Pass
--db-prefix	Database Prefix
--db-port	Database Port
--site-name	commands.site.install.options.site-name
--site-mail	commands.site.install.options.site-mail
--account-name	commands.site.install.options.account-name
--account-mail	commands.site.install.options.account-mail
--account-pass	commands.site.install.options.account-pass
--force	commands.site.install.options.force

Available arguments

Argument	Details
profile	Drupal Profile to be installed

Examples

- Install a drupal project specifying installation type, language code, database configuration, site name, site email and admin credential settings

```
drupal site:install  standard  \
--langcode="en"  \
--db-type="mysql"  \
--db-host="127.0.0.1"  \
--db-name="drupal8"  \
--db-user="u53rn4m3"  \
--db-pass="dbp455"  \
--db-port="3306"  \
--site-name="Drupal 8"  \
--site-mail="admin@example.com"  \
--account-name="admin"  \
--account-mail="admin@example.com"  \
--account-pass="p455w0rd"
```

site:maintenance

Switch site into maintenance mode

Usage:

```
drupal site:maintenance [arguments]
sma
```

Available arguments

Argument	Details
mode	commands.site.maintenance.arguments.mode[on/off]

Examples

- Switch on maintenance

  ```
  drupal site:maintenance  on
  ```

- Switch off maintenance

  ```
  drupal site:maintenance  off
  ```

site:mode

Switch system performance configuration

Usage:

```
drupal site:mode [arguments]
smo
```

Available arguments

Argument	Details
environment	Environment name [dev, prod]

Examples

- Switch system to prod

  ```
  drupal site:mode  prod
  ```

- Switch system to dev

  ```
  drupal site:mode  dev
  ```

site:new

Download a new Drupal project

Usage:

```
drupal site:new [options]
```

Available options

Option	Details
--repository	repository
--directory	directory

site:statistics

Show the current statistics of website.

Usage:

```
drupal site:statistics
sst
```

site:status

View current Drupal Installation status

Usage:

```
drupal site:status [options]
ss
```

Available options

Option	Details
--format	commands.site.status.options.format

Examples

- Get drupal installation status specifying the output format as table

  ```
  drupal site:status  \
  --format="table"
  ```

state:delete

Delete State

Usage:

```
drupal state:delete [arguments]
std
```

Available arguments

Argument	Details
name	State name.

Examples

- Delete state specifying the state name

  ```
  drupal state:delete  comment.maintain_entity_statistics
  ```

state:override

Override a State key.

Usage:

```
drupal state:override [arguments]
sto
```

Available arguments

Argument	Details
key	The State key to override.
value	The State value to set.

Examples

- Override state value specifying the state name and the new value

  ```
  drupal state:override  comment.node_comment_statistics_scale "!!float 1"
  ```

taxonomy:term:delete

Delete taxonomy terms from a vocabulary

Usage:

```
drupal taxonomy:term:delete [arguments]
ttd
```

Available arguments

Argument	Details
vid	

Examples

- Delete all terms of the "tags" vocabulary

  ```
  drupal taxonomy:term:delete  tags
  ```

test:run

Run Test unit from tests available for application

Usage:

```
drupal test:run [arguments] [options]
ter
tr
```

Available options

Option	Details
--url	commands.test.run.arguments.url

Available arguments

Argument	Details
test-class	Test Class
test-methods	Test method(s) to be run

theme:download

Download theme in application

Usage:

```
drupal theme:download [arguments] [options]
thd
td
```

Available options

Option	Details
--composer	Use --composer option for manage the theme download with Composer

Available arguments

Argument	Details
theme	the Theme name
version	Theme version i.e 1.x-dev

Examples

- Download theme specifying name and version

  ```
  drupal theme:download  Alina 7.x-1.2
  ```

theme:install

Install theme or themes in the application

Usage:

```
drupal theme:install [arguments] [options]
thi
ti
```

Available options

Option	Details
--set-default	Set theme as default theme

Available arguments

Argument	Details
theme	commands.theme.install.options.module

Examples

- Install theme specifying the name

  ```
  drupal theme:install  mytheme
  ```

theme:path

Returns the relative path to the theme (or absolute path)

Usage:

```
drupal theme:path [arguments] [options]
thp
```

Available options

Option	Details
--absolute	Return theme absolute path

Available arguments

Argument	Details
theme	Theme name

Examples

- Get the path of mytheme

  ```
  drupal theme:path  mytheme
  ```

theme:uninstall

Uninstall theme or themes in the application

Usage:

```
drupal theme:uninstall [arguments]
thu
tu
```

Available arguments

Argument	Details
theme	commands.theme.uninstall.options.module

Examples

- Uninstall theme specifying the name

  ```
  drupal theme:uninstall  mytheme
  ```

translation:cleanup

commands.translation.cleanup.description

Usage:

```
drupal translation:cleanup [arguments]
tc
```

Available arguments

Argument	Details
language	commands.translation.cleanup.arguments.language
library	commands.translation.cleanup.arguments.library

translation:pending

commands.translation.pending.description

Usage:

```
drupal translation:pending [arguments] [options]
tp
```

Available options

Option	Details
--file	commands.translation.pending.options.file

Available arguments

Argument	Details
language	commands.translation.pending.arguments.language
library	commands.translation.pending.arguments.library

translation:stats

commands.translation.stats.description

Usage:

```
drupal translation:stats [arguments] [options]
ts
```

Available options

Option	Details
--format	commands.translation.stats.options.format

Available arguments

Argument	Details
language	commands.translation.stats.arguments.language
library	commands.translation.stats.arguments.library

translation:sync

commands.translation.sync.description

Usage:

```
drupal translation:sync [arguments] [options]
tsy
```

Available options

Option	Details
--file	commands.translation.stats.options.file

Available arguments

Argument	Details
language	commands.translation.sync.arguments.language
library	commands.translation.sync.arguments.library

update:entities

Applying Entity Updates

Usage:

```
drupal update:entities
upe
```

Examples

- Update entities

  ```
  drupal update:entities
  ```

update:execute

Execute a specific Update N function in a module, or execute all

Usage:

```
drupal update:execute [arguments]
upex
upe
```

Available arguments

Argument	Details
module	The Module name.
update-n	Specific Update N function to be executed

Examples

- Update all entities

  ```
  drupal update:execute
  ```

user:create

Create users for the application

Usage:

```
drupal user:create [arguments] [options]
uc
```

Available options

Option	Details
--roles	User roles
--email	User email
--status	User status

Available arguments

Argument	Details
username	User name to be created
password	User password

Examples

- Create user specifying username, password, role, email and status

```
drupal user:create  john p455w0rd  \
--roles='authenticated'  \
--email="john@anexusit.com"  \
--status="1"
```

- Create admin user specifying username, password, role, email and status

```
drupal user:create  doe p455w0rd  \
--roles='administrator'  \
--email="doe@anexusit.com"  \
--status="1"
```

user:delete

Delete users for the application

Usage:

```
drupal user:delete [options]
ud
```

Available options

Option	Details
--user-id	User id to be deleted
--roles	Roles associated to users to be deleted

Examples

- Delete user specifying the id and the user role

  ```
  drupal user:delete  \
  --user-id="2"
  --roles='authenticated'
  ```

- Delete user specifying its id

  ```
  drupal user:delete  \
  --user-id="3"
  ```

user:login:clear:attempts

Clear failed login attempts for an account.

Usage:

```
drupal user:login:clear:attempts [arguments]
ulca
uslca
```

Available arguments

Argument	Details
uid	User ID.

user:login:url

Returns a one-time user login url.

Usage:

```
drupal user:login:url [arguments]
ulu
uslu
```

Available arguments

Argument	Details
user-id	User ID.

Examples

- Get one time login url for user id 1

  ```
  drupal user:login:url  1
  ```

user:password:hash

Generate a hash from a plaintext password.

Usage:

```
drupal user:password:hash [arguments]
uph
usph
```

Available arguments

Argument	Details
password	Password(s) in text format

Examples

- Get hash of the word "p455w0rd"

  ```
  drupal user:password:hash  p455w0rd
  ```

user:password:reset

Reset password for a specific user.

Usage:

```
drupal user:password:reset [arguments]
upr
uspr
```

Available arguments

Argument	Details
user	User ID
password	Password in text format

Examples

- Update password specifying the user id and the new password

  ```
  drupal user:password:reset  2 p455w0rd
  ```

user:role

Adds/removes a role for a given user

Usage:

```
drupal user:role [arguments]
ur
```

Available arguments

Argument	Details
operation	commands.user.role.operation
user	commands.user.role.user
role	commands.user.role.role

Examples

- Add administrator role to the user admin specifying the username and the role

  ```
  drupal user:role  add admin administrator
  ```

- Remove administrator role from the user admin specifying the username and the role

  ```
  drupal user:role  remove admin administrator
  ```

views:disable

Disable a View

Usage:

```
drupal views:disable [arguments]
vd
vdi
```

Available arguments

Argument	Details
view-id	commands.views.debug.arguments.view-id

Examples

- Disable content view

  ```
  drupal views:disable  content
  ```

- Disable frontpage view

  ```
  drupal views:disable  frontpage
  ```

views:enable

Enable a View

Usage:

```
drupal views:enable [arguments]
ve
```

Available arguments

Argument	Details
view-id	commands.views.debug.arguments.view-id

Examples

- Enable content view

  ```
  drupal views:enable  content
  ```

- Enable frontpage view

  ```
  drupal views:enable  frontpage
  ```

FAQ (Frequently Asked Questions) about DrupalConsole

Having some trouble with DrupalConsole? These questions and answers are a good place to start. FAQs are categorized into the following types of issues:

Installation problems

When you run DrupalConsole from your Drupal 8 root directory, you can get different error messages, we will try to compile the reported issues and how to have them fixed.

Error message:

```
[PDOException] SQLSTATE[HY000] [2002] No such file or directory
```

You will need to edit your 'host' in your 'settings.php' file.

Navigate to `sites/default/settings.php`. In your `settings.php` file, change the `host` to read:

```
'host' => '127.0.0.1'
```

or if your 'settings.php' file already reads:

```
'host' => '127.0.0.1'
```

change it to read:

```
'host' => 'localhost'.
```

After you make the change, be sure to save the file and then run DrupalConsole again.

Error message:

```
[PDOException]
SQLSTATE[HY000] [2002] Can't connect to local MySQL server through socket '/tmp/mysql.soc
```

Creating a symlink pointing to `/tmp/mysql.sock`:

```
ln -s /path/to/your/mysql/data/mysql.sock /tmp/mysql.sock
```

Error message:

```
Fatal error: require(): Failed opening required 'drupal.php'
```

This can be caused by the ioncube loader extension, which can be used to encode and decode PHP files. This extension prevents normal working of any phar files with require/include calls. You must disable the extension.

Warning message:

```
The configuration date.timezone was missing and overwritten with America/Tijuana.
```

Your timezone is not set in php.ini; you must correct this by editing the appropriate php.ini for the command line (there's a separate php.ini for the CLI).

Run `php --ini` and look for "Loaded Configuration File". For example, in Ubuntu:

```
Loaded Configuration File:         /etc/php5/cli/php.ini
```

Edit that file and look for

```
;date.timezone =
```

Uncomment this line and assign the desired timezone as seen on http://php.net/manual/en/timezones.php.

Permissions

To be added.

Commands not listed

This document is a work-in-progress. At any time, it is possible that the Drupal Console project is ahead of the documentation. While we endeavor to keep this book up-to-date, it is always possible that some commands have been created for the Drupal Console, but are not yet described in this document. For a full list of supported commands, use the **list** command, e.g. `drupal list`

If you see a command that is not yet described here, you are also welcome to contribute to this documentation, using the **--help** output for the command as a simple starting point.

Interactive Mode

To be added.

References

Drupal Console code repository

https://github.com/hechoendrupal/drupal-console

Documentation repository

https://github.com/hechoendrupal/drupal-console-book

Resources

- Symfony Components
- Drupal 8
- PHP the right way
- KNP University
- Build a module
- DrupalizeMe
- Git
- Composer
- Box