Table of Contents

- Drupal Console Documentation 0
- What is the Drupal Console 1
  - Why should you care about? 1.1
  - How does Drupal Console help? 1.2
  - Where do I find the project? 1.3
- Getting the project 2
  - Using Composer 2.1
  - Global executable aka Launcher 2.2
  - Installing on Windows 2.3
- Using the project 3
  - How to copy configuration files 3.1
  - How to download, install and serve Drupal 8 3.2
  - How to use Drupal Console in a multi-site installation 3.3
  - How to use Drupal Console in a remote installation 3.4
- Creating custom Commands 4
  - Registering Commands Automatically 4.1
  - Configuring the Command 4.2
  - Command Lifecycle 4.3
  - Getting Services from the Service Container 4.4
- Contributing new features 5
  - Project requirements 5.1
  - Getting the project 5.2
  - Running the project 5.3
  - Keeping your fork up to date 5.4
  - Creating issues and pull requests 5.5
  - Contribute to this documentation 5.6
- Available commands 6
  - about 6.1
  - chain 6.2
  - check 6.3

help	6.4
init	6.5
list	6.6
self-update	6.7
server	6.8
breakpoints:debug	6.9
cache:context:debug	6.10
cache:rebuild	6.11
chain:debug	6.12
config:debug	6.13
config:delete	6.14
config:diff	6.15
config:edit	6.16
config:export	6.17
config:export:content:type	6.18
config:export:view	6.19
config:import	6.20
config:import:single	6.21
config:override	6.22
config:settings:debug	6.23
container:debug	6.24
create:comments	6.25
create:nodes	6.26
create:terms	6.27
create:users	6.28
create:vocabularies	6.29
cron:debug	6.30
cron:execute	6.31
cron:release	6.32
database:client	6.33
database:connect	6.34
database:drop	6.35
database:dump	6.36
database:log:clear	6.37

database:log:debug	6.38
database:restore	6.39
database:table:debug	6.40
devel:dumper	6.41
event:debug	6.42
generate:authentication:provider	6.43
generate:breakpoint	6.44
generate:command	6.45
generate:controller	6.46
generate:doc:cheatsheet	6.47
generate:doc:dash	6.48
generate:doc:data	6.49
generate:doc:gitbook	6.50
generate:entity:bundle	6.51
generate:entity:config	6.52
generate:entity:content	6.53
generate:event:subscriber	6.54
generate:form	6.55
generate:form:alter	6.56
generate:form:config	6.57
generate:help	6.58
generate:module	6.59
generate:module:file	6.60
generate:permissions	6.61
generate:plugin:block	6.62
generate:plugin:ckeditorbutton	6.63
generate:plugin:condition	6.64
generate:plugin:field	6.65
generate:plugin:fieldformatter	6.66
generate:plugin:fieldtype	6.67
generate:plugin:fieldwidget	6.68
generate:plugin:imageeffect	6.69
generate:plugin:imageformatter	6.70

generate:plugin:mail	6.71
generate:plugin:rest:resource	6.72
generate:plugin:rulesaction	6.73
generate:plugin:skeleton	6.74
generate:plugin:type:annotation	6.75
generate:plugin:type:yaml	6.76
generate:plugin:views:field	6.77
generate:post:update	6.78
generate:profile	6.79
generate:routesubscriber	6.80
generate:service	6.81
generate:theme	6.82
generate:twig:extension	6.83
generate:update	6.84
image:styles:debug	6.85
image:styles:flush	6.86
libraries:debug	6.87
locale:language:add	6.88
locale:language:delete	6.89
locale:translation:status	6.90
migrate:debug	6.91
migrate:execute	6.92
module:debug	6.93
module:download	6.94
module:install	6.95
module:path	6.96
module:uninstall	6.97
module:update	6.98
multisite:debug	6.99
multisite:new	6.100
node:access:rebuild	6.101
plugin:debug	6.102
queue:debug	6.103
queue:run	6.104

rest:debug	6.105
rest:disable	6.106
rest:enable	6.107
router:debug	6.108
router:rebuild	6.109
settings:debug	6.110
settings:set	6.111
site:debug	6.112
site:import:local	6.113
site:install	6.114
site:maintenance	6.115
site:mode	6.116
site:new	6.117
site:statistics	6.118
site:status	6.119
state:debug	6.120
state:delete	6.121
state:override	6.122
test:debug	6.123
test:run	6.124
theme:debug	6.125
theme:download	6.126
theme:install	6.127
theme:path	6.128
theme:uninstall	6.129
translation:cleanup	6.130
translation:pending	6.131
translation:stats	6.132
translation:sync	6.133
update:debug	6.134
update:entities	6.135
update:execute	6.136
user:debug	6.137

- user:delete 6.138
- user:login:clear:attempts 6.139
- user:login:url 6.140
- user:password:hash 6.141
- user:password:reset 6.142
- user:role 6.143
- views:debug 6.144
- views:disable 6.145
- views:enable 6.146
- views:plugins:debug 6.147
- yaml:diff 6.148
- yaml:merge 6.149
- yaml:split 6.150
- yaml:update:key 6.151
- yaml:update:value 6.152

FAQ (Frequently Asked Questions) about Drupal Console 7

- Installation problems 7.1
- Permissions 7.2
- Commands not listed 7.3
- Interactive Mode 7.4

References 8

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.

Supporting organizations

What is the Drupal Console?

The Drupal Console is a suite of tools run from a command line interface (CLI) to generate boilerplate code and interact with a Drupal 8 installation. From the ground up, it has been built to utilize the same modern PHP practices which were introduced in Drupal 8.

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.**
- Starting with RC17 there are no longer any commands available that can be run outside of a drupal project. Commands are still available outside of your drupal root. For example, if your drupal root is in a /web subdirectory of your project the `drupal` command can still be run in the root of your project, where your composer.json file and vendor directory exist.

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

```
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
```

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

On Windows there are two ways to install drupal console. One uses Git Bash, the other uses a Windows command prompt. I recommend using the Git Bash utility from the Git for Windows (previously msysgit) program package, since this is the only way you can use drupal console without prefixing it with php.

Using on Git Bash:

If you use Drupal Console on Git Bash, please install packages below:

- Git for Windows
- Composer
- PHP For Windows
- sqlite-tools-win32-x86

Update PATH environment

After installation, you have to include php.exe and sqlite3.exe in your PATH environment variable. For example, if you extracted "PHP For Windows" into "C:\php", and extracted "sqlite-tools-win32-x86" into "C:\sqlite", you can set PATH environment variable as below from command prompt.

```
SETX /M PATH "%PATH%;C:\php;C:\sqlite"
```

Setup php.ini

Drupal Console require some extensions. please enable these extensions in your php.ini.

```
extension=php_gd2.dll
extension=php_pdo_sqlite.dll
extension=php_curl.dll
extension=php_openssl.dll
```

We recommend to enable the following extensions to enable you to use your own language.

```
extension=php_intl.dll
extension=php_mbstring.dll
```

Define certificate

put certificate information provided by Git for Windows.

```
curl.cainfo = C:\Program Files\Git\usr\ssl\certs\ca-bundle.crt;
```

Install Drupal Console globally using composer:

```
$ composer global require drupal/console:@stable
```

You can now execute console using:

```
$ drupal
```

or execute one of the chain available, to execute a quick install execute the following command

```
$ drupal chain --file="C:\Users\username\.console\chain\quick-start.yml"
```

NOTE: You have to provide "Windows-style" path for `file` option.

Using the project

Drupal Console provides two types of commands, `stand alone` and `container aware` commands.

Stand alone commands: These commands can run outside of a Drupal 8 site root.

Container aware 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 existing Drupal 8 sites and have installed the project globally using the instructions in 2.1 or 2.3 you will still need to install it into the Drupal site using the command 'composer require drupal/console:~1.0 --prefer-dist --optimize-autoloader'.

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 `~/.console/` directory. Overriding values on these copied files is how you can change DrupalConsole behaviour.

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

Which files are copied when executing the `init` command.

```
~/.console/
 ├── aliases.yml
 ├── chain
 │   ├── create-data.yml
 │   ├── form-sample.yml
 │   ├── quick-start-mysql.yml
 │   ├── quick-start.yml
 │   ├── sample.yml
 │   ├── site-drop-restore.yml
 │   ├── site-install.yml
 │   └── update-gitbook.yml
 ├── commands.yml
 ├── config.yml
 ├── console.rc
 ├── drupal.fish
 ├── phpcheck.yml
 ├── router.php
 ├── site.mode.yml
 └── sites
     └── sample.yml
```

How to download, install and serve Drupal 8

The easiest way to try Drupal 8 in your local machine is by executing the `chain` command and pass the option `--file=~/.console/chain/quick-start.yml` as shown on the following example.

```
$ drupal chain --file=~/.console/chain/quick-start.yml
```

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

```
commands:
  - command: site:new
    arguments:
      directory: drupal8.dev
      version: 8.0.2
  - command: site:install
    options:
        langcode: en
        db-type: sqlite
        db-file: sites/default/files/.ht.sqlite
        site-name: 'Drupal 8 Quick Start'
        site-mail: admin@example.com
        account-name: admin
        account-mail: admin@example.com
        account-pass: admin
        generate-inline: true
    arguments:
        profile: standard
  - command: server
```

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 `multisite:debug` command to debug multi-site installations and the `--uri` option to interact with multi-site installations.

How to list all known multi sites

```
$ drupal multisite:debug

+---------------------+--------------------------------+
| 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
```

How to use Drupal Console in a remote site installation

Drupal Console allows you to run commands on your local server but actually execute them on a remote server.

You can take advantage of this feature, using the `--target` option and passing the remote site name you want to interact with.

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

Setting up your local computer to use a remote site requires a little configuration.

Edit global configuration

You can provide global configuration to remote connections at the copied file `~/.console/config.yml`. This information is grouped within the `remote` key.

```
application:
  ...
  remote:
    user: root
    port: 22
    console: /usr/local/bin/drupal
    options:
    arguments:
    keys:
      public: ~/.ssh/id_rsa.pub
      private: ~/.ssh/id_rsa
      passphrase: ~/.ssh/passphrase.txt
```

Edit specific site configuration

You can provide specific site configuration by duplicating the copied site file at `~/.console/sites/sample.yml` with a new name at `~/.console/sites/`.

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

Debug sites.

You can list all known local and remote sites by executing the `site:debug` command.

```
$ drupal site:debug

+--------------------+-----------------+------------------------+
| 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  |
+--------------------+-----------------+------------------------+
```

You can show the site configuration details by passing the site name as argument to the `site:debug` command.

```
$ drupal site:debug sample.dev

user: drupal
port: 22
console: /usr/local/bin/drupal
options:
arguments:
keys:
    public: ~/.ssh/id_rsa.pub
    private: ~/.ssh/id_rsa
    passphrase: ~/.ssh/passphrase.txt
root: /var/www/html/drupal
host: 140.211.10.62
remote: true
```

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.

Creating Custom Commands

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

Executing the command using the interactive command questions:

```
 $ drupal generate:command

 // Welcome to the Drupal Command generator
 Enter the module name [devel]:
 > 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) [yes]:
 >

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

Generated or updated files
 Site path: /var/www/drupal.dev
 1 - modules/custom/example/src/Command/DefaultCommand.php
```

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

```
 $ drupal generate:command  --module=example --class=DefaultCommand --name=example:default
```

This command execution will generate a new Command class containing the boiler-plate required to register a new command within your Drupal module.

Registering Commands Automatically

To make the console commands available automatically within a Drupal installation, you will need to:

- Create a `Command` directory inside your module i.e. `src/Command` and create a `PHP` file suffixed with `Command.php` i.e. `MyCustomCommand.php` for each command that you want to provide.
- Make sure you class extends one of `Command` or `ContainerAwareCommand` classes.

Drupal Console provides two types of commands, `stand alone` and `container aware` commands.

Stand alone commands

These commands are listed and can run outside of a Drupal installation, you defined one by extending the `Command` class.

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

class MyStandAloneCommand extends Command
{
}
```

Container aware commands

These commands are listed and must be run against a Drupal, you defined one by extending the `ContainerAwareCommand` class.

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

class MyContainerAwareCommand extends ContainerAwareCommand
{
}
```

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

By using `ContainerAwareCommand` as the base class for the command (instead of the more basic `Command`), you have access to the service container. In other words, you have access to any configured service using the provided `getService` method.

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

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
```

Download Drupal 8

The Drupal Console project only supports Drupal 8; which you will need to download and install locally.

Download Drupal

```
$ drupal site:new drupal8.dev 8.0.0
$ cd drupal8.dev
```

Install Drupal using MySQL:

```
$ drupal site:install standard --langcode=en --db-type=mysql --db-host=127.0.0.1
  --db-name=drupal --db-user=root --db-pass=root --db-port=3306
  --site-name="Drupal 8 Site Install" --site-mail=admin@example.com
  --account-name=admin --account-mail=admin@example.com --account-pass=admin -n
```

Install Drupal using SQLite:

```
$ drupal site:install standard --langcode=en --db-type=sqlite
  --db-file=sites/default/files/.ht.sqlite --site-name="Drupal 8 Site Install"
  --site-mail=admin@example.com --account-name=admin --account-mail=admin@example.com
  --account-pass=admin -n
```

Start the PHP's built in server

```
$ drupal server
```

NOTE: Make sure you use your own user and database credentials when running `site:install` and never user root on production. In this example code, we accept all interactive questions, i.e. answering *“yes”* when passing the `-y` argument.

Getting the project

Fork

Fork your own copy of the Console repository to your account

Clone

Get a copy of your recently cloned version of console in your machine.

```
$ git clone git@github.com:[your-github-user-here]/drupal-console.git
```

Install dependencies

Now that you have cloned the project, you need to download dependencies via Composer.

```
$ cd /path/to/drupal-console
$ composer install
```

Running the project

After using Composer to download dependencies, you can run the project by executing:

```
$ bin/drupal
```

Create a symbolic link

You can run this command to easily access the Drupal Console from anywhere on your system:

```
$ sudo ln -s /path/to/drupal-console/bin/drupal /usr/local/bin/drupal
```

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

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
about	Display basic information about Drupal Console project
chain	Chain command execution
check	System requirement checker
help	Displays help for a command
init	Copy configuration files to user home directory.
list	Lists all available commands
self-update	Update project to the latest version.
server	Runs PHP built-in web server
breakpoints	
breakpoints:debug	Displays breakpoints available in application
cache	
cache:context:debug	Displays current cache context for the application.
cache:rebuild	Rebuild and clear all site caches.
chain	
chain:debug	List available chain files.
config	
config:debug	Show the current configuration.
config:delete	Delete configuration
config:diff	Ouput configuration items that are different in active configuration compared with a directory.
config:edit	Edit the selected configuration.
config:export	Export current application configuration.
config:export:content:type	Export a specific content type and their fields.
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 the selected configuration.
config:override	Override config value in active configuration.
config:settings:debug	Displays current key:value on settings file.
container	
container:debug	Displays current services for an application.
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:debug	List of modules implementing a cron
cron:execute	Execute cron implementations by module or execute all crons
cron:release	Release cron system lock to run cron again
database	
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:debug	Display current log events for the application
database:restore	Restore structure and contents of a database.
database:table:debug	Show all tables in a given database.
devel	
devel:dumper	Change the devel dumper plugin
event	
event:debug	Display current events
generate	

generate:authentication:provider	Generate an Authentication Provider
generate:breakpoint	Generate breakpoint
generate:command	Generate commands for the console.
generate:controller	Generate & Register a controller
generate:doc:cheatsheet	Generate a printable cheatsheet for Commands
generate:doc:dash	Generate the DrupalConsole.docset package for Dash
generate:doc:data	Generate documentations for Commands.
generate:doc:gitbook	Generate documentations for Commands
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 "FormBase"
generate:form:alter	Generate an implementation of hook_form_alter() or hook_form_FORM_ID_alter
generate:form:config	Generate a new "ConfigFormBase"
generate:help	Generate an implementation of hook_help()
generate:module	Generate a module.
generate:module:file	Generate a .module file
generate:permissions	Generate module permissions
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:rest:resource	Generate plugin rest resource

generate:plugin:rulesaction	Generate a plugin rule action
generate:plugin:skeleton	Generate an implementation of a skeleton plugin for those plugins Drupal Console do not have a specific generator
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:debug	List image styles on the site
image:styles:flush	Execute flush function by image style or execute all flush images styles
libraries	
libraries:debug	Displays libraries available in application
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:debug	Display current migration available for the application
migrate:execute	Execute a migration available for application
module	
module:debug	Display current modules available for application
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:debug	List all multisites available in system
multisite:new	Sets up the files for a new multisite install.
node	
node:access:rebuild	Rebuild node access permissions. Rebuilding will remove all privileges to content and replace them with permissions based on the current modules and settings.
plugin	
plugin:debug	Display all plugin types, plugin instances of a specific type, or the definition for a specific plugin.
queue	
queue:debug	Display the queues of your application
queue:run	Process the selected queue.
rest	
rest:debug	Display current rest resource for the application
rest:disable	Disable a rest resource for the application
rest:enable	Enable a rest resource for the application
router	
router:debug	Displays current routes for the application
router:rebuild	Rebuild routes for the application
settings	
settings:debug	List user Drupal Console settings.
settings:set	Change a specific setting value in DrupalConsole config file
site	
site:debug	List all known local and remote sites.
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	Create a new Drupal project
site:statistics	Show the current statistics of website.
site:status	View current Drupal Installation status
state	
state:debug	Show the current State keys.
state:delete	Delete State
state:override	Override a State key.
test	
test:debug	List Test Units available for the application.
test:run	Run Test unit from tests available for application
theme	
theme:debug	Displays current themes for the application
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	Clean up translation files
translation:pending	Determine pending translation string in a language or a specific file in a language
translation:stats	Generate translate stats
translation:sync	Sync translation files
update	
update:debug	Display current updates available for the application
update:entities	Applying Entity Updates
update:execute	Execute a specific Update N function in a module, or execute all
user	
user:debug	Displays current 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:debug	Display current views resources for the application
views:disable	Disable a View
views:enable	Enable a View
views:plugins:debug	Display current views plugins for the application
yaml	
yaml:diff	Compare two YAML files in order to find differences between them.
yaml:merge	Merge two or more YAML files in a new YAML file. Latest values are preserved.
yaml:split	Split a YAML file using indent as separator criteria
yaml:update:key	Replace a YAML key in a YAML file.
yaml:update:value	Update a value for a specific key in a YAML file.

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
--no-debug	Switches off debug mode
--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

Display 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	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
--no-debug	Switches off debug mode
--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

check

System requirement checker

Usage:

```
$ drupal check
```

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 to user home directory.

Usage:

```
$ drupal init [options]
```

Available options

Option	Details
--override	Override configurations files

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

self-update

Update project to the latest version.

Usage:

```
$ drupal self-update [options]
```

Available options

Option	Details
--major	Update to a new major version, if available.
--manifest	Override the manifest file path.
--current-version	Override the version to update from.

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
  ```

breakpoints:debug

Displays breakpoints available in application

Usage:

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

Available arguments

Argument	Details
group	Enter Breakpoint Group Name

cache:context:debug

Displays current cache context for the application.

Usage:

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

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
  ```

chain:debug

List available chain files.

Usage:

```
$ drupal chain:debug
```

config:debug

Show the current configuration.

Usage:

```
$ drupal config:debug [arguments]
$ cde
```

Available arguments

Argument	Details
name	Configuration name.

config:delete

Delete configuration

Usage:

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

Available arguments

Argument	Details
type	Configuration type.
name	Configuration name.

config:diff

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

Usage:

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

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.

config:edit

Edit the selected configuration.

Usage:

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

Available arguments

Argument	Details
config-name	Configuration name.
editor	Editor.

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.

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

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

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.

config:import:single

Import the selected configuration.

Usage:

```
$ drupal config:import:single [arguments]
$ cis
```

Available arguments

Argument	Details
name	Configuration name
file	Path to the import file

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:settings:debug

Displays current key:value on settings file.

Usage:

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

container:debug

Displays current services for an application.

Usage:

```
$ drupal container:debug [arguments]
$ cod
```

Available arguments

Argument	Details
service	commands.container.debug.options.cache

create:comments

Create dummy comments for your Drupal 8 application.

Usage:

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

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

create:nodes

Create dummy nodes for your Drupal 8 application.

Usage:

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

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

Available arguments

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

create:terms

Create dummy terms for your Drupal 8 application.

Usage:

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

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

create:users

Create dummy users for your Drupal 8 application.

Usage:

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

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

create:vocabularies

Create dummy vocabularies for your Drupal 8 application.

Usage:

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

Available options

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

cron:debug

List of modules implementing a cron

Usage:

```
$ drupal cron:debug
```

cron:execute

Execute cron implementations by module or execute all crons

Usage:

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

Available arguments

Argument	Details
module	The Module name.

cron:release

Release cron system lock to run cron again

Usage:

```
$ drupal cron:release
$ crr
```

database:client

Launch a DB client if it's available

Usage:

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

Available arguments

Argument	Details
database	Database key from settings.php

database:connect

Shows DB connection

Usage:

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

Available arguments

Argument	Details
database	Database key from settings.php

database:drop

Drop all tables in a given database.

Usage:

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

Available arguments

Argument	Details
database	Database key from settings.php

database:dump

Dump structure and contents of a database

Usage:

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

Available options

Option	Details
--file	commands.database.dump.option.file

Available arguments

Argument	Details
database	Database key from settings.php

database:log:clear

Remove events from DBLog table, filters are available

Usage:

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

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

database:log:debug

Display current log events for the application

Usage:

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

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

Available arguments

Argument	Details
event-id	DBLog event ID

database:restore

Restore structure and contents of a database.

Usage:

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

Available options

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

Available arguments

Argument	Details
database	Database key from settings.php

database:table:debug

Show all tables in a given database.

Usage:

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

Available options

Option	Details
--database	Database key from settings.php

Available arguments

Argument	Details
table	Table to debug

devel:dumper

Change the devel dumper plugin

Usage:

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

Available arguments

Argument	Details
dumper	Name of the devel dumper plugin

event:debug

Display current events

Usage:

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

Available arguments

Argument	Details
event	Event to debug

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

generate:breakpoint

Generate breakpoint

Usage:

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

Available options

Option	Details
--theme	Theme name
--breakpoints	Breakpoints

generate:command

Generate commands for the console.

Usage:

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

Available options

Option	Details
--module	The Module name.
--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

generate:controller

Generate & Register a controller

Usage:

```
$ drupal generate:controller [options]
$ 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

generate:doc:cheatsheet

Generate a printable cheatsheet for Commands

Usage:

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

Available options

Option	Details
--path	The path to generate the pdf for the documentation
--wkhtmltopdf	The path for the wkhtmltopdf binary in your system

generate:doc:dash

Generate the DrupalConsole.docset package for Dash

Usage:

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

Available options

Option	Details
--path	The path to the directory where the docset will be saved.

generate:doc:data

Generate documentations for Commands.

Usage:

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

Available options

Option	Details
--file	The file to render the command data

generate:doc:gitbook

Generate documentations for Commands

Usage:

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

Available options

Option	Details
--path	The path to render the documentation
--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
--no-debug	Switches off debug mode
--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

generate:entity:config

Generate a new config entity

Usage:

```
$ drupal generate:entity:config [options]
$ 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

generate:entity:content

Generate a new content entity

Usage:

```
$ drupal generate:entity:content [options]
$ 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

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.

generate:form

Generate a new "FormBase"

Usage:

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

Available options

Option	Details
--module	The Module name.
--class	The form class name
--form-id	The Form id
--services	Load services from the container.
--inputs	Create inputs in a form.
--path	Enter the form path
--menu_link_gen	commands.generate.form.options.menu_link_gen
--menu_link_title	commands.generate.form.options.menu_link_title
--menu_parent	commands.generate.form.options.menu_parent
--menu_link_desc	commands.generate.form.options.menu_link_desc

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.

generate:form:config

Generate a new "ConfigFormBase"

Usage:

```
$ drupal generate:form:config [options]
$ 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.
--inputs	Create inputs in a form.
--path	Enter the form path
--menu_link_gen	commands.generate.form.options.menu_link_gen
--menu_link_title	commands.generate.form.options.menu_link_title
--menu_parent	commands.generate.form.options.menu_parent
--menu_link_desc	commands.generate.form.options.menu_link_desc

generate:help

Generate an implementation of hook_help()

Usage:

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

Available options

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

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)

generate:module:file

Generate a .module file

Usage:

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

Available options

Option	Details
--module	The Module name.

generate:permissions

Generate module permissions

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.

generate:plugin:ckeditorbutton

Generate CKEditor button plugin.

Usage:

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

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.

generate:plugin:condition

Generate a plugin condition.

Usage:

```
$ drupal generate:plugin:condition [options]
$ 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)

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

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

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

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

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

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

generate:plugin:mail

Generate a plugin mail

Usage:

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

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.

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

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

generate:plugin:skeleton

Generate an implementation of a skeleton plugin for those plugins Drupal Console do not have a specific generator

Usage:

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

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.

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

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

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

generate:post:update

commands.generate.post:update.description

Usage:

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

Available options

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

generate:profile

Generate a profile.

Usage:

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

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)
--distribution	The distribution name

generate:routesubscriber

Generate a RouteSubscriber

Usage:

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

Available options

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

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
--services	Load services from the container.
--path_service	Path

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
--base-theme	Base theme (i.e. classy, stable)
--regions	Regions
--breakpoints	Breakpoints

generate:twig:extension

Generate a Twig extension.

Usage:

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

Available options

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

generate:update

Generate an implementation of hook_update_N()

Usage:

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

Available options

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

image:styles:debug

List image styles on the site

Usage:

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

image:styles:flush

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

Usage:

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

Available arguments

Argument	Details
styles	The Images Styles name.

libraries:debug

Displays libraries available in application

Usage:

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

Available arguments

Argument	Details
group	Enter Libraries Name

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:debug

Display current migration available for the application

Usage:

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

Available arguments

Argument	Details
tag	Migrate tag

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

Available arguments

Argument	Details
migration-ids	Migration id(s)

module:debug

Display current modules available for application

Usage:

```
$ drupal module:debug [arguments] [options]
$ mod
```

Available options

Option	Details	
--status	Module status [enabled	disabled]
--type	Module type [core	no-core]

Available arguments

Argument	Details
module	commands.module.debug.module

module:download

Download module or modules in application

Usage:

```
$ drupal module:download [arguments] [options]
$ 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

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

Available arguments

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

module:path

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

Usage:

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

Available options

Option	Details
--absolute	Return module absolute path

Available arguments

Argument	Details
module	The Module name (machine name)

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)

module:update

Update core, module or modules in the application

Usage:

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

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.

multisite:debug

List all multisites available in system

Usage:

```
$ drupal multisite:debug
```

multisite:new

Sets up the files for a new multisite install.

Usage:

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

Available options

Option	Details
--site-uri	Site URI to add to sites.php.
--copy-install	Copies existing site from the default install.

Available arguments

Argument	Details
sites-subdir	Name of directory under 'sites' which should be created.

node:access:rebuild

Rebuild node access permissions. Rebuilding will remove all privileges to content and replace them with permissions based on the current modules and settings.

Usage:

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

Available options

Option	Details
--batch	Process in batch mode.

Examples

- Rebuild node access permissions

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

plugin:debug

Display all plugin types, plugin instances of a specific type, or the definition for a specific plugin.

Usage:

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

Available arguments

Argument	Details
type	Plugin type
id	Plugin ID

queue:debug

Display the queues of your application

Usage:

```
$ drupal queue:debug
```

queue:run

Process the selected queue.

Usage:

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

Available arguments

Argument	Details
name	Queue name.

rest:debug

Display current rest resource for the application

Usage:

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

Available options

Option	Details	
--authorization	Rest resource status enabled	disabled

Available arguments

Argument	Details
resource-id	Rest ID

rest:disable

Disable a rest resource for the application

Usage:

```
$ drupal rest:disable [arguments]
$ 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:debug

Displays current routes for the application

Usage:

```
$ drupal router:debug [arguments]
$ rod
```

Available arguments

Argument	Details
route-name	Route names

router:rebuild

Rebuild routes for the application

Usage:

```
$ drupal router:rebuild
$ ror
```

settings:debug

List user Drupal Console settings.

Usage:

```
$ drupal settings:debug
```

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	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
--no-debug	Switches off debug mode
--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
setting-name	Setting name in yaml flatten format to set a value in Drupal Console config file
setting-value	Setting value to set in Drupal Console config file

site:debug

List all known local and remote sites.

Usage:

```
$ drupal site:debug [arguments]
$ sd
```

Available arguments

Argument	Details
target	commands.site.debug.options.target

site:import:local

Import/Configure an existing local Drupal project

Usage:

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

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

site:install

Install a Drupal project

Usage:

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

Available options

Option	Details
--langcode	Drupal language
--db-type	Drupal Database type to be used in install
--db-file	Drupal Database file to be used in install
--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	Drupal site name
--site-mail	Drupal site mail
--account-name	Drupal administrator account name
--account-mail	Drupal administrator account mail
--account-pass	Drupal administrator account password

Available arguments

Argument	Details
profile	Drupal Profile to be installed

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]

site:mode

Switch system performance configuration

Usage:

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

Available arguments

Argument	Details
environment	Environment name [dev, prod]

site:new

Create a new Drupal project

Usage:

```
$ drupal site:new [arguments] [options]
$ sn
```

Available options

Option	Details
--latest	Use this option to select automatically the latest version
--composer	Install Drupal with Composer
--unstable	Use this option to download unstable releases. If not used, you only can install stable releases. Do not use this with latest nor version.

Available arguments

Argument	Details
directory	Directory where to install Drupal
version	Drupal version to download

site:statistics

Show the current statistics of website.

Usage:

```
$ drupal site:statistics
```

site:status

View current Drupal Installation status

Usage:

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

Available options

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

state:debug

Show the current State keys.

Usage:

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

Available arguments

Argument	Details
key	The State key to debug.

state:delete

Delete State

Usage:

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

Available arguments

Argument	Details
name	State name.

state:override

Override a State key.

Usage:

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

Available arguments

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

test:debug

List Test Units available for the application.

Usage:

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

Available options

Option	Details
--test-class	Test Class

Available arguments

Argument	Details
group	Group

test:run

Run Test unit from tests available for application

Usage:

```
$ drupal test:run [arguments] [options]
$ 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:debug

Displays current themes for the application

Usage:

```
$ drupal theme:debug [arguments]
$ tde
```

Available arguments

Argument	Details
theme	Specific theme to debug

theme:download

Download theme in application

Usage:

```
$ drupal theme:download [arguments] [options]
$ 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

theme:install

Install theme or themes in the application

Usage:

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

Available options

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

Available arguments

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

theme:path

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

Usage:

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

Available options

Option	Details
--absolute	Return theme absolute path

Available arguments

Argument	Details
module	The Theme name (machine name)

theme:uninstall

Uninstall theme or themes in the application

Usage:

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

Available arguments

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

translation:cleanup

Clean up translation files

Usage:

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

Available arguments

Argument	Details
language	Language to clean up files against English

translation:pending

Determine pending translation string in a language or a specific file in a language

Usage:

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

Available options

Option	Details
--file	Specific file to determine pending translations against English

Available arguments

Argument	Details
language	Language to determine pending translations against English

translation:stats

Generate translate stats

Usage:

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

Available options

Option	Details	
--format	Define output format table	markdown

Available arguments

Argument	Details
language	Language to generate translation stats against English

translation:sync

Sync translation files

Usage:

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

Available options

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

Available arguments

Argument	Details
language	Language to syncronize against English source files

update:debug

Display current updates available for the application

Usage:

```
$ drupal update:debug
$ upd
```

update:entities

Applying Entity Updates

Usage:

```
$ drupal update:entities
```

update:execute

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

Usage:

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

Available arguments

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

user:debug

Displays current users for the application

Usage:

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

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

user:delete

Delete users for the application

Usage:

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

Available options

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

user:login:clear:attempts

Clear failed login attempts for an account.

Usage:

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

Available arguments

Argument	Details
uid	User ID.

user:login:url

Returns a one-time user login url.

Usage:

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

Available arguments

Argument	Details
user-id	User ID.

user:password:hash

Generate a hash from a plaintext password.

Usage:

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

Available arguments

Argument	Details
password	Password(s) in text format

user:password:reset

Reset password for a specific user.

Usage:

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

Available arguments

Argument	Details
user	User ID
password	Password in text format

user:role

Adds/removes a role for a given user

Usage:

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

Available arguments

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

views:debug

Display current views resources for the application

Usage:

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

Available options

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

Available arguments

Argument	Details
view-id	View ID

views:disable

Disable a View

Usage:

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

Available arguments

Argument	Details
view-id	View ID

views:enable

Enable a View

Usage:

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

Available arguments

Argument	Details
view-id	View ID

views:plugins:debug

Display current views plugins for the application

Usage:

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

Available arguments

Argument	Details
type	Filter views plugins by type

yaml:diff

Compare two YAML files in order to find differences between them.

Usage:

```
$ drupal yaml:diff [arguments] [options]
$ yd
```

Available options

Option	Details
--stats	Print statistics about YAML files comparation
--negate	Define mode diff or equal comparation, possible values TRUE/FALSE or 0/1
--limit	Limit results to a specific number
--offset	Starting point of a limit

Available arguments

Argument	Details
yaml-left	YAML file used as base to compare
yaml-right	YAML file used to find missing parts or differences with the base YAML file

yaml:merge

Merge two or more YAML files in a new YAML file. Latest values are preserved.

Usage:

```
$ drupal yaml:merge [arguments]
$ ym
```

Available arguments

Argument	Details
yaml-destination	Path of new YAML file to store the result of merge.
yaml-files	Path of YAML files to merge

yaml:split

Split a YAML file using indent as separator criteria

Usage:

```
$ drupal yaml:split [arguments] [options]
$ ys
```

Available options

Option	Details
--indent-level	Split YAML file using a specific indent level
--file-output-prefix	commands.yaml.split.options.file-output-prefix
--file-output-suffix	commands.yaml.split.options.file-output-suffix
--starting-key	YAML Key from where start split - useful to extract partial elements
--exclude-parents-key	Exclude the "parents" key from the generated file

Available arguments

Argument	Details
yaml-file	commands.yaml.split.value.arguments.yaml-file

yaml:update:key

Replace a YAML key in a YAML file.

Usage:

```
$ drupal yaml:update:key [arguments]
$ yu
```

Available arguments

Argument	Details
yaml-file	Path of YAML file to update
yaml-key	YAML key to update
yaml-new-key	commands.yaml.update.value.arguments.yaml-new-key

yaml:update:value

Update a value for a specific key in a YAML file.

Usage:

```
$ drupal yaml:update:value [arguments]
$ yuv
```

Available arguments

Argument	Details
yaml-file	Path of YAML file to update
yaml-key	YAML key to update
yaml-value	YAML value to update

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
- Permissions
- Commands not listed
- Interactive Mode

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