Drupal Developer Implementation Manual

A Comprehensive Technical Guide for Enterprise-Grade Development and Deployment

Mastering Drupal Development: A Complete Technical Learning Guide

# Purpose

* The purpose of this manual is to serve as a comprehensive and user-friendly guide for individuals and organizations using Drupal, a flexible and open-source content management system (CMS). This manual aims to simplify the learning curve by providing clear instructions, step-by-step procedures, and practical examples for both new and intermediate users.
* It covers the essential aspects of Drupal, including installation, site structure, content creation, user management, configuration, and customization. By using this guide, users will gain the knowledge needed to efficiently build and maintain dynamic, secure, and scalable websites.
* In addition, this manual provides guidance on using contributed modules and themes, best practices for site performance and security, and tips for troubleshooting common issues. Whether you're a site administrator, content editor, or developer, this manual is designed to help you make the most of Drupal's powerful features.
* Our goal is to empower users to confidently manage their websites, reduce reliance on technical support, and encourage innovation through the effective use of Drupal’s tools and capabilities.

# **Installation and Setup of Drupal**

## **System Requirements**

* PHP 8.1.x or 8.2.x
* MySQL 5.7 or MySQL 8.0
* Apache 2.4.x
* Composer (dependency manager for PHP)

1. **Download Drupal**
   * Use Composer:

composer create-project drupal/recommended-project my\_site\_name\_dir

1. **Set Up Environment**
   * Configure Apache to the /web directory.
   * Create a database for Drupal using phpMyAdmin or CLI.
2. **Run Drupal Installation**  
     
   Open the website URL in a browser.
   * Choose the installation profile (Standard is recommended).
   * Follow on-screen instructions to:
     + Select language
     + Configure database
     + Create site admin account
3. **Post-Installation**
   * Secure settings.php and files folder permissions.
   * Enable required modules from the **Extend** section.

# Login Process

### **Drupal Admin Login Process**

To access the administrative features and manage site content, the user must log in with the provided **administrator credentials**.

1. Navigate to the website’s landing page.
2. Click on the **“Login”** button located at the top right corner of the page.
3. The login form will appear. Enter your **Admin Username**, **Password**, and complete the **CAPTCHA** (if enabled).
4. Click the **“Log in”** button.
5. Upon successful login, you will be redirected to the landing page or the admin dashboard, depending on the site configuration.

Once logged in as an administrator, you will have access to content management, user roles, configuration settings, modules, themes, and other administrative tools.

### **Change Password Process**

It is a standard security practice to change your password periodically. Follow the steps below to change your password:

1. **Log in** to the website using your **admin credentials**.
2. After logging in, click on your **username** at the top of the page, then select **“Edit profile”** from the dropdown menu.
3. On the **Edit profile** page, enter your **current password**, then type your **new desired password** in the appropriate fields.
4. Click the **“Save”** button to update your password.

After saving, your password will be successfully changed. Be sure to remember the new password for future logins.

* + A success message will appear at the top once the user has successfully modified the credential.

# Content Management

## Generic Content Addition

* If the user wants to add any permitted content for their respective website, they need to hover the mouse over "Content," which will be in the top left corner appearing under the black bar menu.
* All the allowed content types will be shown upon clicking "Add Content," and the user can easily add the relevant content from the list of content types.

**Important Notes** :

* If a user wants to add any permitted content, they can select a specific content type from the list and fill in only the relevant fields.
* After entering all required details, click the **“Save” button** at the bottom of the form. If the form is filled correctly, a success message will appear at the top of the page. The content will then be added successfully and can be viewed under its respective section.
* **Content Ordering:** Most content items on the site are displayed in descending order based on the last updated date. This means the most recently added or updated items appear at the top. In some content types, a “Weight” field may be available to manually adjust the display order.
* **Archival Functionality:** The system automatically moves content to the archive page once its expiry date has passed. There is no manual checkbox for archiving. This ensures that only current content is shown on the main site, while expired content is preserved in the archive for reference. Content Management Options: Users with appropriate permissions can add, edit, update, or delete content using the content interface. This ensures that the published information remains accurate, up-to-date, and well-maintained.

# Generic Content Search, Addition/Modification, Content Translation

### Content Search

Go to the "Content Search" to know the details of any content. The following

activities can be performed through the "Content Search" page:

* The user can search most of the content or filter for specific content types. Any page title appearing at the top of the page can be searched through this filter. The user can also filter their data by selecting the type of content.
* The user can edit or update the content.
* The user can translate the content.
* User can see the details of content like content type, publishing status, content created date, content last updated date, etc.

# **Theme Setup and Frontend**

### **Installing a Theme**

1. Download a contributed theme or create a custom one:
   * Example using Composer:

composer require drupal/bootstrap

1. Go to **Appearance** and install the theme.
2. Set as default.

# **Custom Theme Structure**

* Located at: /themes/custom/[theme\_name]
* Key files:
  + theme.info.yml – theme metadata
  + page.html.twig, node.html.twig – custom Twig templates
  + css/, js/, templates/ – for styling and structure

# **Frontend Development**

* Use Twig templating engine
* Override templates by placing them in the templates folder
* Add CSS/JS in libraries.yml and attach using info.yml

# **Block Configuration**

### **Creating and Placing Blocks**

1. Go to **Structure > Block layout**
2. Select a region and click **Place block**
3. Use either:
   * Custom block (create content directly)
   * Existing blocks (e.g., search, user login)

### **Custom Blocks**

* Go to **Structure > Block layout > Custom block library**
* Add a new custom block with desired content
* Assign to a region and configure visibility settings (e.g., by content type or role)

# **Views**

### **Purpose**

Views allow dynamic display of content, such as listing articles, creating filters, or exposing search pages.

### **Creating a View**

1. Navigate to **Structure > Views > Add view**
2. Provide:
   * View name
   * Content type to display
   * Display format (e.g., grid, table, teaser)
   * Create a page or block
3. Configure filters, sorting, and exposed filters

### **Advanced Features**

* Relationships (e.g., author info)
* Contextual filters (e.g., URL arguments)
* Custom templates (e.g., views-view--[view-name].html.twig)

# **Menu Structure**

### **Creating Menus**

1. Go to **Structure > Menus**
2. Create a new menu or edit existing ones (Main navigation, Footer menu)

### **Adding Menu Links**

* Add content or custom URLs
* Define menu item title and parent item
* Assign weight for order

### **Menu Display**

* Use blocks to display menus in regions

# **Content Types**

### **Creating a Content Type**

1. Go to **Structure > Content types > Add content type**
2. Define:
   * Name and description
   * Submission form settings
   * Publishing options

### **Adding Fields**

* Add custom fields (text, image, reference, etc.)
* Use **Field UI** to configure widgets and display.

### **Display Settings**

* Customize form and display layouts
* Use **Manage Display** to change label visibility, formatter, and layout

# Entity Reference and Relationships

Drupal’s **Entity Reference** field allows you to build **relationships between different content types**, users, taxonomy terms, or any other Drupal entities. This is especially useful when

you want to link one piece of content to another — for example, linking a **News** item to an **Author profile**.

### Why Use Entity References?

* To create structured, reusable content relationships
* Avoid content duplication (e.g., reuse author bios, locations, products)
* Enable **related content blocks or views**
* Facilitate **better content organization and filtering**

## Use Case Example: News → Author Relationship

You want to allow content creators to associate each News article with an Author profile.

### Step 1: Create the Related Content Type

* Go to **Structure → Content types → Add content type**
* Name it Author
* Add fields like: Name, Designation, Bio, Image
* Save and publish a few sample Author entries

### Step 2: Add an Entity Reference Field

1. Go to:  
   **Structure → Content types → News → Manage fields**
2. Click **“Add field”**, choose:
   * **Reference → Content**
   * Label it as Author
3. Under **Reference type**, select:
   * Author content type
4. Optional settings:
   * Allow only one author (Number of values: 1)
   * Enable autocomplete widget for easier selection
   * Limit selection using a **view or taxonomy filter** (advanced)
5. Click **Save field settings** and configure label/required options

### Step 3: Display Author on the News Page

1. Go to:  
   **Manage display** tab of the News content type  
   (/admin/structure/types/manage/news/display)
2. Find the Author field and:
   * Enable it for display
   * Choose display format (e.g., **Rendered entity**, **Label only**, or **Link to content**)
3. Save the display settings

### Step 4: Display Related Content Using Views

You can now show content **related by entity reference** using Views.

#### Example: Show all News Articles by the Same Author

1. Go to **Structure → Views → Add view**
2. View name: News by Author
3. Show Content of type News
4. Create a block display

#### Add a Contextual Filter:

* Add Content: Author (entity reference)
* Provide default value: **Content ID from URL**
* This will dynamically filter News based on the selected Author

## Entity Reference Revisions (Advanced)

The **Entity Reference Revisions** module lets you **lock referenced content to a specific revision**. This is useful for **paragraphs, layout builder**, or versioned reference use cases.

* Enable module:

composer require drupal/entity\_reference\_revisions

* Use it when referencing **Paragraphs**, **Layout Sections**, or if you need revision tracking of nested/embedded content.

# **Custom Module Development in Drupal 10**

### Overview

In Drupal 10, custom modules allow developers to extend or customize the functionality of a Drupal site. Modules are built following a defined structure and use **PSR-4** auto loading for class loading. This ensures your code is organized, efficient, and follows modern PHP standards.

### **1. Module Architecture**

A custom module in Drupal follows a specific directory structure and naming convention:

web/

└── modules/

└── custom/

└── my\_module/

├── my\_module.info.yml

├── my\_module.routing.yml (optional - for routes)

├── my\_module.module (optional - procedural code)

├── src/

│ └── Controller/

│ └── MyController.php (for controller logic)

└── templates/ (optional - for Twig templates)

* The custom folder is used to keep all user-defined modules separate from core and contributed ones.
* my\_module is your custom module's machine name.

### **2. Create Custom Module Skeleton**

* To create your module, run the following command (from your Drupal project root):
* mkdir -p web/modules/custom/my\_module
* This command creates the directory structure for your custom module.

### **3. Create the .info.yml File**

Every module must include an .info.yml file. This file tells Drupal about the module's name, type, Drupal core compatibility, and any dependencies.

#### my\_module.info.yml (Sample)

* name: 'My Custom Module'
* type: module
* core\_version\_requirement: ^10
* package: Custom
* dependencies:
  + node

**Explanation of fields:**

* **name:** Human-readable name of the module (shown in admin interface).
* **type:** Always module for custom modules.
* **core\_version\_requirement**: Specifies the Drupal core version the module supports (here: Drupal 10).
* **package:** Groups the module under a section in the admin UI (like “Custom”).
* **dependencies:** Lists other modules that must be enabled (e.g., node for working with content types).

### **4. PSR-4 Autoloading**

Drupal 10 uses **PSR-4** to autoload PHP classes. Place your classes inside the src/ directory, and structure your folders to match the namespace.

*For example, a controller class might look like:*

***<?php***

***namespace Drupal\my\_module\Controller;***

***use Drupal\Core\Controller\ControllerBase;***

***class MyController extends ControllerBase {***

***public function content() {***

***return [***

***'#markup' => $this->t('Hello from My Custom Module!'),***

***];***

***}***

***}***

This file would be saved as:

* web/modules/custom/my\_module/src/Controller/MyController.php
* Drupal automatically maps this file based on the PSR-4 namespace.

### **5. Enable Your Module**

Once you've created the .info.yml file, clear the cache:

* drush cr
* Then enable your module:
* drush en my\_module

Or via the admin UI:  
**Admin → Extend → Find "My Custom Module" → Enable**

### Summary

Creating a custom module in Drupal 10 involves:

* Setting up a clean directory structure.
* Defining metadata with a .info.yml file.
* Following PSR-4 auto loading for all PHP classes.
* Optionally creating routes, services, and templates depending on the module’s functionality.

# Introduction to Drupal Hooks

In Drupal, **hooks** are a fundamental concept that allows modules to interact with the Drupal core or other modules without modifying their code directly. Hooks let you “hook into” the system and alter or extend its behavior.

Think of hooks as **event listeners** or **callback functions** triggered by Drupal at specific points in the request lifecycle or when certain actions occur. By implementing these hook functions in your module, you can customize how Drupal behaves.

### How Hooks Work

* Drupal core or contributed modules **invoke** hook functions at various points.
* Your custom module implements these hook functions by following a naming convention.
* When Drupal triggers a hook, it calls your module’s function, allowing your code to run.
* This mechanism supports modular and extensible architecture, keeping the system flexible and maintainable.

## Example: The Help Hook (hook\_help())

One widely used hook is hook\_help(), which allows your module to provide contextual help text on Drupal’s admin pages.

### What is hook\_help()?

* It displays **help information** about your module or specific admin pages.
* The help text appears on the **Help tab** under the module listing or other relevant pages.

### How to Implement hook\_help()?

Suppose your module’s machine name is my\_module. The hook implementation will be:

**function my\_module\_help($route\_name, $route\_match) {**

**switch ($route\_name) {**

**case 'help.page.my\_module':**

**return t('This is the general help page for My Custom Module.');**

**case 'entity.node.add\_form':**

**return t('Here you can create new content related to My Custom Module.');**

**default:**

**return t('Custom help message for developers.');**

**}**

**}**

* $route\_name identifies the current page.
* $route\_match provides additional context.
* The function returns a translatable string using t() for the appropriate page.

### Why Use It?

* Improves usability by giving admins clear guidance.
* Reduces the need for external documentation.
* Supports multilingual sites through translation.

## Common Drupal Hooks

**Here are some important hooks that modules commonly implement:**

| **Hook Name** | **Purpose** |
| --- | --- |
| hook\_help() | Provides help text for your module in admin pages. |
| hook\_form\_alter() | Alters existing forms to add, remove, or modify fields. |
| hook\_menu() | Defines custom menu items and routes (Drupal 7 and earlier). In Drupal 8/9/10 use routing.yml instead. |
| hook\_theme() | Registers custom Twig templates and theme functions. |
| hook\_entity\_insert() | Reacts when an entity (like a node) is created. |
| hook\_entity\_update() | Reacts when an entity is updated. |
| hook\_entity\_delete() | Reacts when an entity is deleted. |
| hook\_block\_info() | Defines new blocks (Drupal 7). In Drupal 8+, use plugins instead. |
| hook\_permission() | Defines custom permissions for your module. |
| hook\_cron() | Implements periodic tasks to be run on cron. |
| hook\_user\_login() | Reacts when a user logs in. |

## Summary

* **Hooks** are the primary way Drupal modules interact with the system and each other.
* They allow customization without hacking core or other modules.
* Implement hooks by defining functions named using your module’s machine name + hook name.
* The **help hook** is an example that provides contextual help.
* Drupal provides many hooks for forms, entities, users, menus, permissions, and more.

# Taxonomy Management

### What is Taxonomy in Drupal?

**Taxonomy** is the system in Drupal used to **classify and organize content** using vocabularies and terms. It helps create structured categories, tags, and labels, enabling better content filtering, navigation, and relationships across your site.

### Key Terms

| **Term** | **Description** |
| --- | --- |
| **Vocabulary** | A group or collection of taxonomy terms (e.g., Tags, Categories) |
| **Term** | An individual item in a vocabulary (e.g., "News", "Events") |
| **Hierarchical** | Terms can be nested under one another (parent/child terms) |

### Creating and Managing Vocabularies

#### Step 1: Navigate to Taxonomy

* Go to:  
  **Structure → Taxonomy**  
  /admin/structure/taxonomy

#### Step 2: Add a Vocabulary

1. Click **“Add vocabulary”**.
2. Enter a **Name** (e.g., “Topics”, “Industries”, “Tags”).
3. Optionally, add a **description** for internal reference.
4. Click **Save**.

#### Step 3: Manage Vocabulary Terms

Once a vocabulary is created:

* Click **“Add terms”** to begin populating it.
* Each **term** can include:
  + Name (required)
  + Description (optional)
  + URL alias
  + Parent term (for hierarchy)

Use the **drag-and-drop interface** to reorder or nest terms under one another.

### Adding and Nesting Terms Hierarchically

Terms can be nested to create **tree-like structures**, such as:

- Services

- Consulting

- Development

- Products

- Software

- Hardware

To do this:

1. While creating or editing a term, select a **Parent term**.
2. Save the term — it will appear nested in the list view.

This structure can later be used for **menu building**, **filters**, or **breadcrumb trails**.

### Using Taxonomy with Content Types

To associate taxonomy with content:

1. Go to **Structure → Content types** and select a content type (e.g., Article).
2. Click **“Manage fields”**.
3. Click **“Add field”** → choose **“Term reference”**.
4. Select the **Vocabulary** to link (e.g., “Tags”).
5. Configure whether users can select one or multiple terms.
6. Save and update form/display settings as needed.

Now, while creating or editing content, users can tag or categorize using the terms.

### Using Taxonomy in Views (Filtering and Display)

One of the most powerful uses of taxonomy is filtering content using **Views**.

#### Example: Show all articles tagged with "Events"

1. Go to **Structure → Views → Add View**
2. Show content of type "Article"
3. Add a **Filter Criteria**:
   * Field = "Tags" (or your term reference field)
   * Operator = "Is one of"
4. Add an **Exposed Filter** to allow users to filter by tag on the frontend.
5. Save the view.

### Benefits of Using Taxonomy

* Organize and classify content logically
* Enable **filters, menus, and navigation**
* Improve **searchability and SEO**
* Reuse across multiple content types
* Create **dynamic displays** with Views and blocks

### Summary Table

| **Feature** | **Description** |
| --- | --- |
| Vocabularies | Collections of terms (like Categories, Tags) |
| Terms | Individual classification items |
| Hierarchical Nesting | Terms can have parent-child relationships |
| Term Reference Field | Attach taxonomy to content types |
| Use in Views | Filter, sort, and display content by taxonomy |

# Form Creation in Drupal

Forms are essential for collecting user input, whether for contact pages, surveys, registrations, or custom data entry. Drupal provides multiple ways to create forms — through contributed modules for quick setup or by building custom forms for complex requirements.

### 1. Creating Forms Using Contributed Modules

Contributed modules simplify form creation without coding. They are ideal for content editors or site builders who want powerful forms with minimal technical effort.

#### Popular Form Modules:

* **Webform**  
  The most popular module for building complex and customizable forms, surveys, polls, and quizzes.  
  **Features:**
  + Drag-and-drop form builder interface
  + Conditional logic and validation
  + Multi-page forms
  + Submission management and export (CSV, Excel)
  + Email notifications and confirmations
* **Contact**  
  Built-in module for simple contact forms per site or per user.

#### Steps to Create a Form Using Webform

1. **Install and enable the Webform module** via Extend page or Composer.
2. Navigate to **Manage → Structure → Webforms** (/admin/structure/webform).
3. Click **“Add webform”**.
4. Enter a name and description for your form.
5. Use the **drag-and-drop interface** to add fields like text, email, checkboxes, selects, dates, files, etc.
6. Configure field properties, validations, and conditional logic as needed.
7. Save the form.
8. Place the webform on a page via blocks or direct URLs.
9. View and export submissions from the Webform dashboard.

### 2. Creating Custom Forms with Code

For developers needing fully tailored forms beyond what contributed modules offer, Drupal provides a Form API to create forms programmatically.

#### Key Concepts:

* Forms are created as **PHP classes** or functions implementing Drupal’s Form API.
* Form elements, validation, submission handlers, and routing are defined in code.
* Forms can be attached to pages, blocks, or custom routes.

#### Basic Example of a Custom Form

**php**

**namespace Drupal\my\_module\Form;**

**use Drupal\Core\Form\FormBase;**

**use Drupal\Core\Form\FormStateInterface;**

**class ExampleForm extends FormBase {**

**// Form ID**

**public function getFormId() {**

**return 'example\_form';**

**}**

**// Build form elements**

**public function buildForm(array $form, FormStateInterface $form\_state) {**

**$form['name'] = [**

**'#type' => 'textfield',**

**'#title' => $this->t('Name'),**

**'#required' => TRUE,**

**];**

**$form['email'] = [**

**'#type' => 'email',**

**'#title' => $this->t('Email Address'),**

**'#required' => TRUE,**

**];**

**$form['submit'] = [**

**'#type' => 'submit',**

**'#value' => $this->t('Submit'),**

**];**

**return $form;**

**}**

**// Validate form inputs**

**public function validateForm(array &$form, FormStateInterface $form\_state) {**

**if (!filter\_var($form\_state->getValue('email'), FILTER\_VALIDATE\_EMAIL)) {**

**$form\_state->setErrorByName('email', $this->t('Invalid email address.'));**

**}**

**}**

**// Handle form submission**

**public function submitForm(array &$form, FormStateInterface $form\_state) {**

**\Drupal::messenger()->addMessage($this->t('Thank you, @name, for your submission.', [**

**'@name' => $form\_state->getValue('name'),**

**]));**

**}**

#### **Steps to Use Custom Form:**

1. Create a custom module.
2. Add the form class file in src/Form/ExampleForm.php.
3. Define a route in your module’s .routing.yml to display the form.
4. Clear cache and visit the route URL to see the form in action.

# Custom Permissions for Fields

In Drupal, permissions are usually assigned at the **content type level** (e.g., who can edit or view a node). But sometimes, you may want **fine-grained control** — such as showing or hiding **specific fields** based on user roles. This is where the **Field Permissions module** comes in.

### Why Use Field-Level Permissions?

* Hide **internal notes** from regular users or authors
* Show **admin-only fields** (e.g., moderation notes, status flags)
* Let certain roles **edit**, **view**, or **not access** specific fields

## Use Case Examples

| **Use Case** | **Description** |
| --- | --- |
| Internal Comments | Editors can see internal notes, viewers cannot |
| Status Fields | Admin can mark as “Reviewed”, users can't edit it |
| Hidden Metadata | Show custom metadata only to staff or admins |

## Step-by-Step: Setting Up Field-Level Permissions

### Step 1: Install the Field Permissions Module

* Using Composer:

composer require drupal/field\_permissions

* drush en field\_permissions -y

Or through Admin UI:

* Go to **Extend** (/admin/modules)
* Enable **Field Permissions**

### Step 2: Add a Field with Permissions

1. Go to:  
   **Structure → Content types → [Your Type] → Manage fields**
2. Click **Add field**, choose the desired field type  
   (e.g., **Text (plain)** for internal notes)
3. Under **Field Settings**, you’ll now see **Field visibility options**:

Check the following as needed:

* + **Users may view this field**
  + **Users may edit this field**

1. After saving, go to:  
   **People → Roles → [Your Role] → Permissions**
2. Under “Field Permissions” section:
   * Assign **view/edit** access for specific fields by role

### Step 3: Verify Behavior

1. Log in as different roles (e.g., Editor vs. Authenticated User)
2. Visit a node with the custom field
3. Check whether visibility and editability match expectations

## Tips & Notes

* These permissions apply to **node forms** and **display** of fields.
* Combine with **View Modes** and **Field Display settings** for flexible control.
* You can use **Custom roles** to create layered access (e.g., Reviewer, Auditor, Author).

## Summary Table

| **Task** | **Path/Command** |
| --- | --- |
| Enable module | composer require drupal/field\_permissions drush en field\_permissions -y |
| Set field-level options | Manage fields → Field settings |
| Assign field permissions | People → Roles → Permissions |
| Test role-based visibility | Login as multiple roles |

## Bonus: Combine with Content Access Control

For even finer control:

* Use **Content Access** or **Permissions by Term** for content-wide access
* Use **Field Permissions** for field-specific visibility/editing

# Caching in Drupal – Simplified Guide

### Why Caching is Important

Caching in Drupal helps **improve performance** and **reduce server load** by storing and reusing previously generated content. It makes your site faster and more scalable, especially under heavy traffic.

### Key Caching Layers in Drupal

Drupal has multiple layers of caching to handle different types of content:

1. **Render Cache**
   * Stores individual pieces of content (like blocks, views).
   * Helps avoid rebuilding parts of the page each time.
2. **Dynamic Page Cache**
   * Stores entire pages for anonymous users (non-logged-in).
   * Speeds up page loads by serving cached versions instead of rebuilding from scratch.

### How to Enable Caching via Drush

You can enable caching layers using Drush (Drupal's command-line tool):

* drush cache:enable render
* drush cache:enable dynamic\_page\_cache

### Use Redis or Memcached (Advanced)

For better performance, you can store cache data in memory using tools like **Redis** or **Memcached**.

#### **Example: Using Redis**

1. Install the **Redis contributed module**.
2. Add this line in your settings.php:

**$settings['cache']['default'] = 'cache.backend.redis';**

Redis will now handle your caching, making it faster and more efficient than using the default database backend.

## Clear Cache When Needed

Whenever you make changes to your site—like updating code, modifying configuration, installing a new module, or altering themes—you need to **clear the cache**. This ensures Drupal recognizes the changes and displays the updated content or behavior.

### Option 1: Using Drush (Command Line)

If you have access to the command line, use this simple command:

***drush cr***

* cr stands for **cache-rebuild**.
* This command clears **all Drupal caches** and rebuilds them immediately.
* It’s fast and ideal for developers.

### Option 2: Using the Admin Interface

If you prefer the graphical interface:

1. Log in to the **Drupal Admin Panel**.
2. Go to:  
   **Configuration** → **Performance**  
   (/admin/config/development/performance)
3. Click the **“Clear all caches”** button.

This will clear and rebuild all caches, similar to the Drush command.

### Summary

* Use drush cr for quick command-line cache rebuilds.
* Or use the **“Clear all caches”** button from the admin panel under **Configuration > Performance**.
* Always clear the cache after making system-level changes to ensure your site runs properly.
* Clear cache with drush cr after updates.

# **Multilingual & Localization Framework**

Required for global products — enables content translation, interface localization, and language negotiation.

### **Enable Language Modules**

drush en language content\_translation interface\_translation

### **Add Languages**

* Navigate to **Configuration → Regional and language → Languages** → Add language
* Configure fallback, negotiation method, and translation workflows.

# **Media Management and Responsive Image Configuration**

Centralized digital asset handling with media entities and responsive display via breakpoints.

**Enable Media Modules**

drush en media media\_library responsive\_image

**Set Responsive Styles**

* Use **Configuration → Media → Image styles** and **Breakpoint groups**

# Security Hardening and Audit Protocols

Enforces OWASP-based hardening and proactive vulnerability mitigation.

**Disable Unused Permissions**

* Audit all roles to ensure no excessive permissions are granted.

**Enforce HTTPS**

* Redirect HTTP to HTTPS in .htaccess config.
* Set config['https'] = TRUE; in settings.php

## Security Kit (seckit) – Drupal Security Hardening Module

### What is Security Kit?

The **Security Kit (seckit)** module helps **protect your Drupal site** from common web vulnerabilities by letting you configure HTTP security headers and behaviors that strengthen your site’s defenses.

It focuses on **client-side security** — stopping things like:

* Cross-Site Scripting (**XSS**)
* Clickjacking
* MIME-type sniffing
* Content injection attacks

### How to Install and Enable Security Kit

#### Step 1: Install the module (via Composer, if available):

composer require drupal/seckit

#### **Step 2: Enable the module:**

drush en seckit -y

Or enable it from the **Admin Interface**:

* Go to **Extend** (/admin/modules)
* Search for **Security Kit**
* Check the box and click **Install**

### Configure Security Kit

Once enabled:

1. Go to:  
   **Configuration** → **System** → **Security Kit**  
   (/admin/config/system/seckit)
2. From here, you can configure:
   * **X-Frame-Options**: Prevents clickjacking (e.g., set to SAMEORIGIN)
   * **Content Security Policy (CSP)**: Restrict what scripts and resources can load
   * **X-XSS-Protection**: Helps browsers prevent reflected XSS attacks
   * **X-Content-Type-Options**: Prevents MIME-type sniffing
   * **Referrer Policy**: Controls how much referrer information is sent
   * **Strict Transport Security (HSTS)**: Forces HTTPS if configured
3. Click **Save Configuration** after making changes.

### 

### Best Practices

* Start with default settings, then gradually apply stricter rules.
* Use **CSP** cautiously; test well to avoid breaking front-end functionality.
* Enable **HSTS** only if your site fully supports HTTPS.

### Summary

| **Feature** | **Purpose** |
| --- | --- |
| X-Frame-Options | Prevents clickjacking |
| X-XSS-Protection | Mitigates reflected XSS |
| X-Content-Type-Options | Prevents MIME sniffing |
| Content-Security-Policy | Limits loaded scripts & resources |
| HSTS | Enforces HTTPS across site |

# URL Aliases and Redirects

Managing your URLs properly is important for **SEO**, **usability**, and **site structure clarity**. Drupal provides tools to create **clean, readable URLs** and manage **redirects** efficiently.

### 1. **Creating User-Friendly URLs (Manual Aliases)**

By default, Drupal creates URLs like:

/node/25

This is not user-friendly or SEO-optimized.

Instead, you can manually set a **URL alias** to something like:

/about-us

#### **How to Create or Edit a URL Alias:**

1. While creating or editing any content (e.g., Page or Article), scroll down to the **“URL Alias”** section.
2. Uncheck **“Generate automatic URL alias”** if it’s checked.
3. Enter your custom URL in the **URL alias** field (e.g., /company/history).
4. Save the content.

**Tip**: Always use lowercase letters, hyphens (-) instead of spaces, and avoid special characters in aliases.

### 2. **Using Pathauto for Automatic Aliases**

Managing aliases manually for every content item can be time-consuming. The **Pathauto** module automatically generates SEO-friendly URLs based on content type, title, taxonomy, etc.

#### Example:

An Article titled **"Tips for Healthy Living"** may get an alias like:

/articles/tips-for-healthy-living

#### How to Use Pathauto:

1. Install the **Pathauto** module (along with its dependency, **Token**).
2. Enable it using:

drush en pathauto token -y

1. Go to:  
   **Configuration → Search and metadata → URL aliases**  
   (/admin/config/search/path)
2. Set **Patterns** for each content type (e.g., articles/[node:title]).
3. Save configuration.

Now, whenever content is created, Drupal will auto-generate a clean URL based on the pattern.

### 3. **Redirecting Old URLs with the Redirect Module**

If you change a page’s URL or move content, it’s important to set up a **redirect** from the old URL to the new one to avoid broken links and preserve SEO ranking.

The **Redirect** module helps with this.

#### How to Set Up Redirects:

1. **Install the Redirect module:**

**drush en redirect -y**

1. **When you change a URL alias, the redirect module automatically creates a redirect from the old alias to the new one (if this setting is enabled).**
2. **To manually add a redirect:**
   * **Go to Configuration → Search and metadata → URL redirects  
     (/admin/config/search/redirect)**
   * **Click Add Redirect.**
   * **Enter the Source path (old URL, e.g., /old-page) and the Redirect to path (e.g., /new-page).**
   * **Save.**

# Cron and Scheduled Tasks

In Drupal, **cron** is a background task system that automatically performs routine maintenance operations such as clearing caches, indexing content for search, checking for updates, and more.

Think of cron as Drupal’s way of keeping the site **healthy, up-to-date, and optimized** behind the scenes.

### How Cron Works in Drupal

Drupal cron runs specific **tasks (functions)** at scheduled intervals. These tasks are defined by the Drupal core or contributed modules.

#### Examples of what cron does:

* Indexing new content for internal search
* Clearing expired cache entries
* Sending out scheduled emails
* Aggregating RSS feeds
* Running maintenance for modules like Scheduler, Pathauto, etc.

Each time cron runs, it triggers all pending tasks in a batch.

### 

### Manual vs. Automatic Cron

#### **Automatic Cron (Default Setting)**

By default, Drupal is configured to run cron **automatically** when users visit the site — based on a time interval set in the admin settings.

This is useful for low-traffic sites, but it's not always reliable for consistent execution.

**Set interval:**

* Go to **Configuration → System → Cron**  
  (/admin/config/system/cron)
* Choose how often cron should run (e.g., every 3 hours)

#### **Manual Cron**

You can manually trigger cron tasks in two ways:

1. **Via Admin Interface:**
   * Go to **Configuration → System → Cron**
   * Click the **“Run cron”** button
2. **Via Drush (recommended for developers):**
   * drush cron
   * This command instantly triggers all cron tasks.

### Using Automated Cron Modules

If your site requires more reliable and customizable cron handling, use one of the following contributed modules:

#### **Automated Cron (Core Feature)**

* Already included in Drupal core.
* Runs based on site traffic.
* Good for small/low-traffic sites.
* Can be configured from /admin/config/system/cron.

#### **Elysia Cron (Advanced Cron Control)**

Elysia Cron is a **contributed module** that offers more control and flexibility than the built-in system.

**Key Features;**

* Set **different run intervals** for each module's cron task
* Disable cron for specific modules
* Logs execution times
* Schedule cron via **system cron jobs** (e.g., crontab on Linux)

**How to use Elysia Cron:**

1. Install the module:

composer require drupal/elysia\_cron

drush en elysia\_cron -y

1. Go to:  
   **Configuration → System → Elysia Cron**  
   (/admin/config/system/cron/settings)
2. Set individual cron schedules per module (e.g., run search indexing every 6 hours, but clear expired sessions every hour)

### Setting Up a System Cron (Recommended for Live Sites)

For high-traffic or production sites, **use your server’s cron service** (Linux crontab, Windows Task Scheduler, etc.) to run Drupal cron regularly and reliably.

### Summary

| **Option** | **Use Case** | **Method** |
| --- | --- | --- |
| Automatic Cron | Basic/low-traffic sites | Triggered by user visits |
| Manual Cron | One-time or on-demand execution | Run via Admin UI or drush cron |
| Elysia Cron | Advanced scheduling per module | Set via config, run via system cron |
| System Cron Job | Reliable execution on production sites | Use OS-level scheduler (e.g., crontab) |

**Best Practice:** For production sites, disable automatic cron and use a system-level cron job (e.g., with Elysia Cron + Drush) for consistent performance.

# Security and Updates

Maintaining strong security is essential to protect your Drupal site from threats like unauthorized access, data loss, or site compromise. This includes keeping software up to date, securing user permissions, and using Drupal's security modules.

### Keeping Core and Modules Up to Date

Outdated modules or core versions are a major security risk.

#### How to check for updates:

* **Go to:  
  Reports → Available updates (/admin/reports/updates)**
* **Click “Check manually” to refresh the list.**
* **Update status is shown for both core and contributed modules/themes.**

#### Update via **Drush**:

composer update drupal/core --with-dependencies

drush updb

Always back up your site before performing updates.

### Best Practices for User and Permission Security

* Follow the **principle of least privilege** — give users only the permissions they need.
* Regularly review:  
  **People → Permissions** (/admin/people/permissions)
* Avoid giving administer site configuration or use PHP access to non-admin users.
* Use **strong passwords**, ideally with enforced password policies (via modules like Password Policy).

# Performance Optimization

Performance tuning improves page load times, reduces server load, and improves user experience — especially on high-traffic sites.

### Enabling Caching Layers

Drupal has built-in caching systems:

1. **Render Cache** – Caches individual content pieces (blocks, entities).
2. **Dynamic Page Cache** – Speeds up page loads for anonymous users.

#### **Enable via Drush:**

**drush cache:enable render**

**drush cache:enable dynamic\_page\_cache**

Or configure at:  
**Configuration → Development → Performance**  
(/admin/config/development/performance)

### Using Redis or Memcached

For improved performance, especially on large sites, use **in-memory caching**:

* **Redis**: Stores cache in memory for fast retrieval.
* Requires the Redis module and PHP Redis extension.

#### Sample in settings.php:

$settings['cache']['default'] = 'cache.backend.redis';

### CSS/JS Aggregation

Combines and minifies CSS and JS files to reduce page load times.

* Enable under:  
  **Configuration → Development → Performance**
* Options:
  + Aggregate CSS files
  + Aggregate JavaScript files

Recommended for production environments.

### Image Styles and Lazy Loading

* Drupal allows creation of **image styles** (thumbnail, medium, large) with automatic resizing and effects.
* Set styles at:  
  **Configuration → Media → Image styles**

#### Lazy loading:

Drupal 10 uses **native lazy loading** by default (loading="lazy"), reducing page weight and improving initial load speed.

# Logging and Error Monitoring

Effective logging helps monitor site behaviour, detect bugs, and catch security threats.

### Viewing Logs

Enable **Database Logging** module (dblog):

drush en dblog -y

Then view logs at:  
**Reports → Recent log messages** (/admin/reports/dblog)

It shows:

* Errors
* Access attempts
* Cron execution
* Module-specific messages

### Syslog (for server-level logging)

**Syslog** sends logs to the operating system log (e.g., /var/log/syslog) — more secure and scalable than DB logging.

Enable the module:

drush en syslog -y

### Setting Up Watchdog Notifications

Use modules like:

* **SMTP** (to send logs via email)
* **Nagios Monitoring** (for external alerts)
* **Loggly / Papertrail** (external log aggregation)

No built-in email alert system exists in core, so combine **Logging + Rules** or external tools to send notifications for errors.

# Content Moderation & Workflows

Drupal supports a **moderation workflow** system that helps manage how content moves through stages — such as **Draft**, **Needs Review**, and **Published**. This is essential for teams that require **editorial review** before publishing content.

### Draft → Review → Publish Lifecycle

By default, when **Content Moderation** is enabled, content can go through the following **states**:

| **State** | **Description** |
| --- | --- |
| **Draft** | Content is saved but not visible to the public |
| **Needs Review** | Content is submitted for approval before publishing |
| **Published** | Content is live and visible to site visitors |
| **Archived** | Content is stored but no longer visible (optional state) |

This allows an **editorial team** to collaboratively work on content — from creation to approval and final publishing.

### How to Enable Content Moderation

1. Enable the core modules:

drush en content\_moderation workflows -y

1. Go to:  
   **Configuration → Workflow → Workflows**  
   (/admin/config/workflow/workflows)
2. Click **“Edit”** on the **Editorial** workflow (default one provided).
3. Assign this workflow to your desired **content types** (e.g., Article, Page).

### Assigning Permissions for Workflow Stages

You can control **who can do what** at each moderation stage by managing user roles and permissions.

Go to:  
**People → Permissions** (/admin/people/permissions)

Look for these key permissions:

* **View own unpublished content**
* **Use editorial transition Draft → Needs Review**
* **Use editorial transition Needs Review → Published**
* **Use editorial transition Published → Archived**
* **Create / edit / delete [Content type]**

Assign them to appropriate roles:

| **Role** | **Typical Permissions** |
| --- | --- |
| **Content Editor** | Create and edit drafts, submit for review |
| **Reviewer** | Approve or reject content |
| **Administrator** | Full access to publish or bypass workflow |

### Setting Up Custom Workflows (Optional)

You can create your own **custom workflows** if the default doesn’t meet your needs.

#### **Steps:**

1. Go to:  
   **Configuration → Workflow → Workflows**
2. Click **“Add Workflow”**
3. Name your workflow (e.g., “Policy Approval Flow”)
4. Add **states** (e.g., "Draft", "Reviewed", "Approved")
5. Add **transitions** between states (e.g., Draft → Reviewed)
6. Assign the workflow to specific **content types** or **entity types**
7. Save and assign **transition permissions** per role

### Content Moderation View (Admin UI)

* You can track moderation status at:  
  **Content → Moderated Content**  
  (/admin/content/moderated)

**This shows:**

* Current state of each content item
* Who last updated it
* Available transitions

# **Backup, Recovery, and Disaster Readiness**

Backup automation and disaster response protocols are vital for SLA compliance.

### **Backup Strategies**

* Use Backup and Migrate module for periodic offsite backups.

# **Search, Indexing, and Faceted Navigation**

Replace core search with **Apache Solr** or **ElasticSearch** for enterprise-grade indexing.

### **Install Search API**

drush en search\_api search\_api\_solr facets

### **Define Index, Attach Views**

* Create custom Views to expose indexed content.
* Enable **facets** for layered navigation.

# **Layout Builder and Component-Based Page Assembly**

Enables non-linear, reusable UI composition with drag-and-drop layout management for editors.

### **Enable Layout Builder**

drush en layout\_builder layout\_discovery

### **Activate per Content Type**

* Go to **Structure → Content types → [Your type] → Manage display**
* Enable **Use Layout Builder**

### **Add Custom Blocks as Layout Components**

* Define custom blocks and embed using Layout UI.

# **Paragraphs Module for Flexible Content Architecture**

Empowers structured content authoring via field able, nest able entity bundles.

# **CKEditor Customization and Content Governance**

Content authors need controlled editing environments with formatting restrictions.

# **Customize Editor Profiles**

* Go to **Configuration → Content authoring → Text formats**
* Assign **CKEditor plugins**, limit HTML tags per role

# **Conceptual Definition and Implementation of Drupal Basic Page Content Type**

The **Basic Page** is an intrinsic, node-based content entity bundle preconfigured in Drupal core. It functions as a **static content container**, primarily used for non-temporal, informational web pages that require minimal field complexity.

### **Technical Characteristics:**

* **Entity Type:** node
* **Bundle:** page
* **Primary Fields:**
  + title (string, required, translatable)
  + body (text with text format, optional)
* **Default Workflow:** Unpublished/published with no complex moderation states (can be extended)
* **URL Alias:** Usually auto-generated by pathauto module using tokens (e.g., /about-us)

### **Lifecycle and Usage Paradigm:**

1. **Instantiation**  
   Creation of a Basic Page entity is executed via the node entity form at /node/add/page, invoking \Drupal\node\Entity\Node::create() with bundle ‘page’.
2. **Storage**  
   Entity data is persisted in the node\_field\_data and node\_field\_revision tables, with field data normalized per Drupal’s Field API schema.
3. **Rendering**  
   The Basic Page renders via Twig templates with overrides possible in:
   * node--page.html.twig
   * page.html.twig (overall page template)  
     Data passed to templates includes processed body content and metadata.
4. **Extensibility**  
   Additional fields and behaviors can be appended via the Field UI or programmatically by defining new field storages attached to the page bundle.
5. **Access Control**  
   Access permissions default to ‘View published content’ for anonymous users; granular control achievable via hook implementations or contributed modules.

### 

### **Pragmatic Developer Notes:**

* Use Basic Page when content structure is simple — no complex entity references or multiple fields required.
* For modular or rich content needs, create custom content types or utilize Paragraphs.
* Always ensure text format policies are consistent with security requirements to prevent XSS.

# Site Configuration Export/Import

### What is Configuration Management in Drupal?

Drupal’s **Configuration Management** system allows you to export, import, and synchronize your site’s configuration settings (such as content types, views, fields, permissions, and more) between different environments (e.g., development, staging, production).

Configuration is stored in human-readable YAML (.yml) files, making it easy to track changes, share settings, and automate deployments.

### Why Use Configuration Export/Import?

* To **migrate configuration** changes from local or development sites to production.
* To maintain consistency across multiple environments.
* To enable version control for site configuration.
* To back up your site’s configuration separately from content.

### Step 1: Access Configuration Synchronization

1. Log in as an administrator.
2. Go to:  
   **Manage → Configuration → Development → Configuration synchronization**  
   (/admin/config/development/configuration)

### Step 2: Exporting Configuration

You can export your entire site configuration or individual items.

* **Export Full Site Configuration**  
  Click **“Export”** → **“Full archive”** to download a ZIP archive containing all configuration .yml files.
* **Export Single Configuration**  
  Click **“Export”** → **“Single item”**, then select the configuration item (e.g., a specific view or content type) to export as YAML.

Save these files safely. They can be committed to version control or transferred to another environment.

### Step 3: Importing Configuration

* Go to **Import** → **Full archive** or **Single item** depending on what you want to import.
* Upload the .yml file(s) you exported from another environment.
* Review the changes and confirm the import.
* Drupal will update the configuration on your site accordingly.

### Step 4: Managing Configuration Splits

Configuration splits let you maintain **different configuration sets** for multiple environments (e.g., disabling certain modules on staging but enabling on production).

* Install the **Config Split** module.
* Define **splits** that apply only in specific environments (development, production).
* Enable or disable configuration items per split.
* This helps avoid environment-specific conflicts when importing/exporting configuration.

### Best Practices

* Always **backup your database and files** before importing configuration.
* Use **version control (Git)** to track changes in your exported configuration files.
* Use **drush commands** for advanced configuration export/import:

bash

CopyEdit

drush config-export # Exports config to config/sync directory

drush config-import # Imports config from config/sync directory

* Coordinate configuration changes carefully when working with multiple developers.

# Metatag and SEO Settings

Search Engine Optimization (SEO) is essential for ensuring your Drupal site ranks well on search engines like Google. One of the most effective tools for this in Drupal is the **Metatag** module, which allows you to control the meta tags (title, description, canonical URL, etc.) for every page of your site.

### Why Use Metatags?

Meta tags are snippets of information placed in a page's HTML <head> section. They don’t appear on the page itself but help **search engines understand** the content and improve how your site is displayed in search results.

**Examples of Metatags:**

* Page title
* Meta description
* Canonical URL
* Open Graph (Facebook)
* Twitter Card tags

## Step-by-Step: Installing and Configuring Metatag Module

### 1. **Install the Metatag Module**

#### Option A: Via Admin Interface

1. Go to **Extend** (/admin/modules)
2. Search for “Metatag”
3. Check the box and click **Install**

#### Option B: Via Composer (recommended)

composer require drupal/metatag

drush en metatag -y

### 2. **Basic Configuration**

1. Go to:  
   **Configuration → Search and Metadata → Metatag**  
   (/admin/config/search/metatag)
2. You’ll see a list of entities and bundles (e.g., Global, Article, Basic Page).
3. Click **Edit** next to:
   * **Global**: sets default tags for all pages
   * A specific content type (e.g., Article): overrides global tags

### 3. **Set Default Meta Tags**

Within a metatag configuration:

* **Page title**:  
  Use a token like [node:title] | [site:name]  
  (This creates a dynamic title like "About Us | My Site")
* **Description**:  
  [node:summary] or custom short description  
  Helps improve click-through rate in search results
* **Canonical URL**:  
  [current-page:url] (prevents duplicate content issues)
* **Advanced Tags** (optional):
  + **Open Graph** for Facebook sharing
  + **Twitter Cards** for optimized Twitter sharing
  + **Robots** tag (e.g., noindex, nofollow)

Click **Save** once you're done.

## Using Tokens for Dynamic Tags

Drupal tokens allow you to insert dynamic values into meta tags, like page titles or summaries.

### Commonly Used Tokens:

| **Token** | **Output Example** |
| --- | --- |
| [node:title] | The content’s title |
| [site:name] | The site’s configured name |
| [node:summary] | Teaser or body summary |
| [current-page:url] | Full canonical URL |
| [date:short] | Current date |

To explore all available tokens, click the **"Browse available tokens"** link while editing a metatag configuration.

## SEO Best Practices for Drupal

1. **Use clean, descriptive page titles**  
   Use dynamic tokens with proper separators like pipes (|) or dashes (-)
2. **Write compelling meta descriptions**  
   Meta descriptions don’t influence ranking directly but affect CTR
3. **Avoid duplicate content with canonical URLs**  
   Always set [current-page:url] as the canonical tag
4. **Enable clean URLs**  
   Drupal handles this by default but verify it’s enabled
5. **Install SEO-supporting modules:**
   * **Redirect**: for 301 URL redirection
   * **Pathauto**: auto-generate clean URLs using tokens
   * **XML Sitemap**: create sitemaps for search engines
   * **Simple XML Sitemap** (alternative to XML Sitemap)
6. **Use Open Graph and Twitter Tags**  
   Improve how pages look when shared on social media
7. **Regularly audit your site** using tools like:
   * [Ahrefs](https://ahrefs.com)
   * Google Search Console
   * Screaming Frog SEO Spider

# Search Engine Sitemap

A **sitemap** is a structured XML file that lists all the important pages of your website to help **search engines crawl** and index your content effectively. In Drupal, this is best managed using the **Simple XML Sitemap** module.

### Why Use a Sitemap?

* Improves **SEO** by helping search engines discover all your pages
* Ensures **new content** is indexed faster
* Helps detect and fix **crawl issues**
* Used by platforms like **Google Search Console** and **Bing Webmaster Tools**

## Installing and Enabling Simple XML Sitemap

### Step 1: Install the Module

Via Composer (recommended):

* composer require drupal/simple\_sitemap
* drush en simple\_sitemap -y

Or via the admin UI:

1. Go to **Extend** (/admin/modules)
2. Search for “Simple XML Sitemap”
3. Enable the module and required submodules (e.g., sitemap generator)

## Basic Configuration

Navigate to:  
**Configuration → Search and Metadata → Simple XML Sitemap**  
/admin/config/search/simplesitemap

### 1. **Generate the Sitemap**

* After enabling, click **“Generate Sitemap”**
* A sitemap file will be created at:

https://example.com/sitemap.xml

This file is automatically updated when content is created or updated.

### 2. **Include/Exclude Content Types, Taxonomies, etc.**

To control what is included in the sitemap:

1. Go to the **Inclusion Settings** tab or visit:  
   /admin/config/search/simplesitemap/settings
2. You can configure sitemap inclusion for:
   * **Content types** (e.g., Article, Page)
   * **Taxonomies**
   * **Menus**
   * **Custom paths**
   * **User profiles or media entities (if needed)**
3. Set:
   * **Priority**: how important this content is (0.1–1.0)
   * **Change Frequency**: how often this content changes (daily, weekly, etc.)
4. Save the configuration.

### **Adding Custom URLs**

If you have custom routes/pages not managed by Drupal entities:

* Go to the **Custom Links** tab
* Add URLs like /custom-path, /landing-page, etc.
* Set priority and change frequency

## Submitting Sitemap to Google Search Console

Once your sitemap is generated and published at /sitemap.xml:

### Step-by-Step:

1. Go to Google Search Console
2. Select your website (domain property)
3. Click on **"Sitemaps"** in the left menu
4. Enter the path:

sitemap.xml

1. Click **Submit**

Google will now regularly crawl your sitemap and index new content.

### Regenerate the Sitemap

To manually refresh the sitemap:

drush sitemaps:generate

Or use the "Regenerate" button on the configuration page.

# Custom Error Pages (403 / 404)

In Drupal, users might encounter errors when trying to access content:

* **403 Access Denied**: When a user lacks permission to view a page.
* **404 Page Not Found**: When a user visits a URL that doesn’t exist.

Instead of showing default error messages, you can create **custom error pages** that are more helpful and user-friendly.

### Why Customize Error Pages?

* Provides a **better user experience**
* Keeps users engaged by offering helpful links or navigation
* Maintains **branding and design consistency**
* Helps with **SEO** (proper handling of errors avoids penalties)

## Step-by-Step: Creating Custom Error Pages

### Step 1: Create the Error Content

1. Go to:  
   **Content → Add content → Basic page** (or any content type)
2. Create a page for:
   * **403 Error**: Example title – “Access Denied”
   * **404 Error**: Example title – “Page Not Found”
3. Add user-friendly messaging like:
   * “You don’t have permission to view this page. Please log in or return to the homepage.”
   * “Oops! We couldn’t find that page. Try searching or go back to the homepage.”
4. Include links to helpful sections of your site (e.g., Home, Contact Us, Sitemap)
5. Publish the pages and note their URLs (e.g., /access-denied, /page-not-found)

### Step 2: Set Error Page URLs in Configuration

1. Navigate to:  
   **Configuration → System → Site Information**  
   (/admin/config/system/site-information)
2. Scroll to the **Error Pages** section:
   * **Default 403 (Access Denied) page** → enter the path you created (e.g., /access-denied)
   * **Default 404 (Not Found) page** → enter the path (e.g., /page-not-found)
3. Save the configuration.

Make sure these pages are publicly accessible (no login required), or they’ll cause a redirect loop.

## Optional: Redirecting Broken URLs

To handle old or broken links more gracefully:

1. Install the **Redirect** module:

bash

CopyEdit

composer require drupal/redirect

drush en redirect -y

1. Go to **Configuration → Search and metadata → URL redirects**  
   (/admin/config/search/redirect)
2. Create redirects from outdated or incorrect URLs to valid pages.

## Best Practices

* Match your **site's theme** and design for consistent branding
* Use **clear and friendly language**
* Add a **search box**, **popular links**, or **contact form**
* Avoid indexing these pages in search engines (add noindex meta tag using Metatag module)

# Site Maintenance Mode

### What is Maintenance Mode?

**Maintenance Mode** in Drupal is a special site state used when the site needs to be temporarily taken offline for updates, development, or troubleshooting. During this mode, regular visitors see a maintenance message instead of the usual website content, while administrators can still access and work on the site.

### Why Use Maintenance Mode?

* To apply core, module, or theme updates safely without affecting users.
* To perform database updates or configuration changes.
* To troubleshoot site issues without public visibility.
* To prevent users from interacting with the site during critical operations.

### How to Enable Maintenance Mode

#### Step 1: Log in as an Administrator

Make sure you have admin privileges to access site configuration.

#### Step 2: Navigate to Maintenance Settings

1. Go to:  
   **Manage → Configuration → Development → Maintenance mode**  
   (/admin/config/development/maintenance)

#### Step 3: Enable Maintenance Mode

* Check the box labeled **“Put site into maintenance mode”**.
* This will temporarily disable public access.

### Step 4: Customize the Maintenance Message

* In the **“Maintenance mode message”** field, you can write a custom message that visitors will see while the site is offline.
* Use this space to inform users about expected downtime, reason for maintenance, or contact information.

Example message:

“Our website is currently undergoing scheduled maintenance. We apologize for the inconvenience and will be back shortly. Thank you for your patience.”

### Step 5: Allow Administrator Access

* By default, users with the **Administrator** role can still log in and access the site during maintenance.
* This lets admins perform updates or troubleshooting while regular visitors see the maintenance message.

### Step 6: Save Configuration

* Click **Save configuration** to apply the maintenance mode settings.

### Disabling Maintenance Mode

Once your updates or maintenance tasks are complete:

1. Return to the **Maintenance mode** settings page.
2. Uncheck the **“Put site into maintenance mode”** box.
3. Save the configuration to make the site live again.

### Additional Tips

* **Use Drush (command line)**: You can enable maintenance mode quickly with the command:
  + drush sset system.maintenance\_mode 1
  + And disable it with:
  + drush sset system.maintenance\_mode 0
* **Clear cache** after enabling or disabling maintenance mode to ensure changes reflect properly:
  + drush cr

# Drupal Admin Toolbar Tabs – Detailed Guide

### **Content**

The **Content** tab is your central hub for managing everything visible on the site.

#### **Features:**

* **Add Content**:  
  Create new pieces of content like:
  + **Article** (blog/news-style posts)
  + **Basic Page** (static content such as About or Contact pages)
  + Any custom content types you’ve created (e.g., Events, Products)
* **Manage Content**:  
  See a complete list of all content on the site. You can:
  + Filter by type, author, published/unpublished
  + Perform bulk actions like publishing, unpublishing, or deleting
  + Quickly access edit, delete, or view links
* **Media (if Media module is enabled)**:
  + Upload and manage images, documents, and other media files
  + Organize media by type or usage

#### Use Cases:

* Updating a blog post
* Deleting outdated pages
* Reviewing draft content before publishing

### **2. Structure**

The **Structure** tab defines how your site is organized, including how content is created, displayed, and navigated.

#### **Features:**

* **Content Types**:
  + Define different types of content (e.g., Blog Post, Product)
  + Add custom fields (e.g., image, date, category)
  + Set display modes (Teaser, Full content)
* **Taxonomy**:
  + Create vocabularies (e.g., Tags, Categories)
  + Organize and filter content by terms
* **Menus**:
  + Edit main navigation, footer, or user menu
  + Add, reorder, and remove links
  + Create custom menus for specific sections
* **Views**:
  + Create dynamic lists (e.g., list of recent blog posts, filtered galleries)
  + Customize sorting, filters, display styles (grid, table, etc.)
  + Show content in blocks, pages, or attachments
* **Blocks**:
  + Place reusable pieces of content in specific regions (e.g., sidebar, footer)
  + Set visibility conditions (show on specific pages, for specific roles)
* **Custom Block Library**:
  + Create reusable block content
  + Assign blocks to multiple regions or pages
* **Webforms (optional module)**:
  + Build custom forms with fields like name, email, dropdowns, checkboxes
  + Configure email notifications and submission settings

#### **Use Cases:**

* Creating a custom content type for job postings
* Designing a category/tag system for articles
* Showing a list of events using Views

**3. Appearance**

The **Appearance** tab lets you control the look and feel of the website.

#### **Features:**

* **Themes**:
  + Install new themes from Drupal.org
  + Set a default theme (for visitors) and admin theme
  + Enable or disable themes as needed
* **Theme Settings**:
  + Add custom logo, favicon, or site name/slogan
  + Customize layout and colors (if supported by the theme)
  + Control display settings for user interface elements

#### **Use Cases**:

* Applying a responsive theme for mobile users
* Adding your organization's logo to the site
* Adjusting theme layout to support wider content

**Reports**

The **Reports** tab provides important diagnostics and logs for site maintenance and troubleshooting.

#### Features:

* **Status Report**:
  + Overview of system health (PHP version, file permissions, security updates)
* **Recent Log Messages**:
  + View system errors, warnings, and notices
  + Useful for debugging site issues
* **Available Updates**:
  + Shows core and contributed module/theme updates
  + Helps keep your site secure and up to date
* **Top '404' Errors**:
  + Identifies broken links and missing pages
* **Fields Report**:
  + Overview of all custom fields across content types and entities
* **Views, Blocks, and Menu Reports**:
  + Lists and statuses of created Views, blocks, and menu items

#### **Use Cases:**

* Checking for module updates
* Troubleshooting a blank or broken page
* Fixing broken links causing 404 errors

### **Configuration**

The **Configuration** tab is where global settings and system behaviors are managed.

#### Categories & Features:

##### **System Settings**:

* **Site Information**: Site name, slogan, email, front page
* **Cron**: Automate tasks like clearing caches, indexing search
* **File System**: Public/private file storage locations

##### **People and Permissions**:

* **Roles and Permissions**: Define what users can see/do
* **Account Settings**: Control user registration and login settings

##### **Content Authoring**:

* **Text Formats and Editors**: Configure filters (e.g., basic HTML, full HTML), WYSIWYG editors
* **Comment Settings**: Enable/disable comments on content types

##### **Search and Metadata**:

* **URL Aliases**: Set custom URLs for content (e.g., /about-us instead of /node/3)
* **Meta Tags (if installed)**: Improve SEO with titles, descriptions
* **Search Pages**: Configure internal site search behavior

##### **Development and Performance**:

* **Caching**: Set page and block cache settings
* **Maintenance Mode**: Temporarily disable access to public users
* **Error Reporting**: Control what errors are shown

#### **Use Cases:**

* Enabling clean URLs for SEO
* Setting the default homepage
* Limiting who can create or edit content

### **People**

Manage all user accounts and permissions.

#### **Features:**

* **User List**: View and manage existing users
* **Add User**: Manually create user accounts
* **Roles**:
  + Create roles like Editor, Manager, Contributor
  + Assign permissions to each role (fine-grained access control)
* **Permissions**: Assign what each role can access (administer content, manage menus, etc.)

#### **Use Cases:**

* Creating a new user for a content editor
* Blocking spam users
* Setting up permissions for an editor to publish content

