

Drupal 7 to Drupal 10 Migration Consultant Cheat Sheet

This guide is designed as a complete consultant and engineer checklist to upgrade any Drupal 7 site to Drupal 10. It includes real-world command references, best practices, validation steps, and troubleshooting guidance. The steps are organized in exact order as executed by professional migration consultants.

1. System Comparison & Preparation

Drupal 7 to Drupal 10 Migration Consultant Cheat Sheet

=====

1. System Comparison

Feature	Drupal 7	Drupal 10
Core	Procedural hooks	Symfony + OOP + services
Themes	PHPTemplate + tpl.php	Twig + libraries.yml
Routing	hook_menu()	routing.yml + controllers
Modules	Pure hooks	Plugins + services + events
DB API	db_query()	Entity API + DBAL
Views	Contrib module	Core module
Panels	Contrib module	Layout Builder (manual)
Blocks	hook_block()	Block Content Entities
Files	file_managed	file_managed
Fields	Field API + collections	Field API + Paragraphs
Paragraphs	Field collections (contrib)	Paragraphs module + Entities
Users	users table	users_field_data + config

2. Install Drupal 10

```
composer create-project drupal/recommended-project mysite
cd mysite
drush site:install
drush en migrate migrate_drupal migrate_drupal_ui -y
```

3. settings.php Configuration

```
$databases['default']['default'] = array (
  'database' => 'drupal10',
  'username' => 'd10_user',
  'password' => 'd10_password',
  'prefix' => '',
  'host' => '127.0.0.1',
  'port' => '',
  'namespace' => 'Drupal\\Core\\Database\\Driver\\mysql',
  'driver' => 'mysql',
);
$databases['migrate']['default'] = array (
  'database' => 'drupal7',
  'username' => 'd7_user',
  'password' => 'd7_password',
  'prefix' => '',
  'host' => '127.0.0.1',
  'port' => '',
  'namespace' => 'Drupal\\Core\\Database\\Driver\\mysql',
  'driver' => 'mysql',
);
```

);

4. Migration Setup and Table Mapping

Drupal 7 Table	Drupal 10 Table
node	node_field_data
node_revisions	node_revision
users	users_field_data
taxonomy_term_data	taxonomy_term_field_data
file_managed	file_managed
comment	comment_field_data
menu_links	menu_link_content_data
field_data_*	field_data_* (fields)

2. Migration Setup & D7 DB Instructions

3. Migration Setup & Table Mappings

Before starting migration:

1. Export your Drupal 7 database using mysqldump or phpMyAdmin.
2. Import the D7 database into your local MySQL/MariaDB server alongside the Dr

Example:

```
mysqldump -u root -p drupal7 > drupal7.sql  
mysql -u root -p drupal7 < drupal7.sql
```

IMPORTANT:

- Do NOT install Drupal 7 site on the same instance.
- The D7 database must exist as a separate DB so Drupal 10 Migrate API can read
- Common database names: drupal7 (old site), drupal10 (new site).

Edit sites/default/settings.php in Drupal 10 to include:

```
$databases['default']['default'] = array (  
  'database' => 'drupal10',  
  'username' => 'd10_user',  
  'password' => 'd10_password',  
  'host' => '127.0.0.1',  
  'driver' => 'mysql',  
);
```

```
$databases['migrate']['default'] = array (  
  'database' => 'drupal7',  
  'username' => 'd7_user',  
  'password' => 'd7_password',  
  'host' => '127.0.0.1',  
  'driver' => 'mysql',  
);
```

Table Name Mapping

Drupal 7 Table	Drupal 10 Table
node	node_field_data
node_revisions	node_revision
users	users_field_data
taxonomy_term_data	taxonomy_term_field_data
file_managed	file_managed
comment	comment_field_data
menu_links	menu_link_content_data
field_data_*	field_data_* (fields)

3. Migration Execution & Verification (with Full Commands)

5. Migration Execution & Command Explanations

`drush migrate-status`

- Lists all available migration definitions.
- Shows current status: Idle, Importing, Stopped, Completed.

`drush migrate-import upgrade_d7_block`

- Imports Drupal 7 blocks to Drupal 10.

`drush migrate-import upgrade_d7_menu`

- Migrates all menus and menu links from D7.

`drush migrate-import upgrade_d7_node`

- Migrates all nodes (content) from Drupal 7.

`drush migrate-import upgrade_d7_taxonomy_term`

- Migrates taxonomy vocabularies and terms.

`drush migrate-import upgrade_d7_user`

- Migrates all user accounts from D7.

`drush migrate-rollback upgrade_d7_node`

- Rolls back node content migration only.

`drush migrate-rollback upgrade_d7_user`

- Rolls back user account migration only.

`drush migrate-import --all`

- Imports all migration groups at once.

`drush cr`

- Clears all Drupal caches.
- Always run after configuration changes or after a migration batch.

mysql verification examples:

`mysql> SELECT COUNT(*) FROM drupal7.node;`

`mysql> SELECT COUNT(*) FROM drupal10.node_field_data;`

`mysql> SELECT COUNT(*) FROM drupal7.users;`

`mysql> SELECT COUNT(*) FROM drupal10.users_field_data;`

`mysql> SELECT COUNT(*) FROM drupal7.file_managed;`

`mysql> SELECT COUNT(*) FROM drupal10.file_managed;`

4. Migration Plugin IDs: upgrade_d7_user, etc.

What are upgrade_d7_user, upgrade_d7_node, etc.?

These are not custom names. They are default migration plugin IDs provided by core's migrate_drupal module for D7 to D10 upgrades.

When a valid Drupal 7 database is connected and migrate_drupal + migrate_drupal_ui are enabled, the system auto-generates these IDs.

You can list them at any time using:

```
drush migrate-status
```

Default IDs and what they migrate:

upgrade_d7_user

Migrates Drupal 7 users (users table → users_field_data).

upgrade_d7_node

Migrates all nodes (all content types).

upgrade_d7_taxonomy_term

Migrates taxonomy terms and vocabularies.

upgrade_d7_file

Migrates managed files (file_managed table).

upgrade_d7_block

Migrates custom blocks.

5. Auto vs Manual Migration

6. What Migrates Automatically vs Manually

Automatically Migrates:

- Content Types
- Nodes
- Users
- Files
- Taxonomy Terms
- Menus
- Basic Fields
- Comments
- Roles & Permissions

Requires Manual Rebuild:

- Views (rebuild at /admin/structure/views)
- Panels (convert to Layout Builder)
- Field Collections (convert to Paragraphs)
- Theme Templates (convert to Twig)
- Custom Contrib Modules (must rewrite)
- Custom Entities (must recreate)

6. Schema Differences & Notes

7. Schema Differences & Migration Notes

Users:

D7: users → D10: users_field_data

Nodes:

D7: node, node_revisions → D10: node_field_data, node_revision

Taxonomy Terms:

D7: taxonomy_term_data → D10: taxonomy_term_field_data

Files:

D7: file_managed → D10: file_managed

Menus:

D7: menu_links → D10: menu_link_content_data

Blocks:

D7: blocks → D10: block_content

Comments:

D7: comment → D10: comment_field_data

Fields:

D7 & D10: field_data_fieldname

Notes:

Entity IDs (uid, nid, fid) remain same if no conflicts.

7. Post-Migration Checks & Expanded Problems

7. Post-Migration Checks & Issues

Check users: /admin/people
Check nodes: /admin/content
Check taxonomy: /admin/structure/taxonomy
Check files: /admin/content/files
Check menus and URL aliases
Run: drush migrate-status
Run: drush cr

9. Common Problems + Solutions

Duplicate IDs:

- Symptom: Migration fails with key conflicts.
- Solution: Always migrate into a clean, empty D10 DB.

Missing files:

- Symptom: Files exist in D7 but not in D10.
- Solution: Manually copy /sites/default/files folder before migration.

Missing migrate plugin:

- Symptom: Drush reports "missing source plugin".
- Solution: Enable migrate_drupal + migrate_drupal_ui modules.

Missing fields after migration:

- Symptom: Some fields not populated.
- Solution: Create field structure manually then rerun drush migrate-import.

Views missing:

- Symptom: Views do not migrate.
- Solution: Recreate manually in D10 Views UI.

Slow performance or memory exhausted:

- Symptom: Large site migrations hang.
- Solution: Break migration into batches: drush migrate-import upgrade_d7_node

Custom entities missing:

- Solution: Rebuild entity types + write custom migrate plugin.

Broken menu links:

- Solution: Manually fix URL paths in D10 menus.

Roles and permissions missing:

- Solution: Verify manually in /admin/people/roles.

Broken URL aliases:

- Solution: Rebuild manually in /admin/config/search/path.

Missing Paragraph fields (Field Collections):

- Solution: Convert Field Collections in D7 to Paragraphs in D10.

User passwords not working:

- Solution: Reset password using D10's user password reset functionality.

Missing file references in entities:

- Solution: Rerun file migrate dependencies: `drush migrate-import upgrade_d7_files`

Multilingual content missing:

- Solution: Verify D10 has multilingual modules enabled before running migrate.

Custom blocks missing:

- Solution: Rebuild custom blocks under `/admin/structure/block`.

Broken redirects:

- Solution: Recreate redirects manually or import via redirect module.

Consultant Pro Tips

- Always test after each migration batch.
- Backup both D7 and D10 databases.
- Never copy `tpl.php` templates into D10.
- Use staging environment before production.
- Estimate 20-30% time for Views & Theme rebuild.

8. How Drupal Migration Batch Processor & Queue Worker Works

Overview:

Drupal Migrate API uses Queue Workers + Batch API to avoid memory overload and timeouts when migrating large datasets.

Triggered by:

1. Web UI (/upgrade) → Batch API
2. CLI (drush) → Queue Worker

Recommended drush batch commands:

```
drush migrate-import upgrade_d7_user --limit=500
drush migrate-import upgrade_d7_file --limit=500
drush migrate-import upgrade_d7_taxonomy_term --limit=500
drush migrate-import upgrade_d7_menu --limit=500
drush migrate-import upgrade_d7_node --limit=500
drush migrate-import upgrade_d7_block --limit=500
```

Check migration status:

```
drush migrate-status
```

Resume stopped migration:

```
drush migrate-import upgrade_d7_node
```

Best Practices:

- Always run large migrations via drush CLI
- Avoid web UI for large imports
- Rerunning migrate-import is safe; completed items are skipped

9. How drush migrate-import triggers Migrate API & Queue Workers

Do you need to write code for Queue Worker?

No, not for standard D7 → D10 migrations.

Drupal core's migrate + migrate_drupal modules provide:

- Queue Worker plugin
- Batch handling via Symfony Queue system
- All default entity types (users, nodes, files, taxonomy, menus, blocks)

You only write custom Queue Workers if:

- You have custom entities
- You build custom migration plugins
- You want to change batch size or custom behavior

99% of migrations never need custom Queue Worker code.

How drush migrate-import works internally

drush migrate-import upgrade_d7_node

↓

Migrate API (MigrateExecutable)

↓

Checks for batch mode (enabled with --limit)

↓

Splits source into batches if large dataset

↓

Queue Worker processes each batch

↓

Each batch loads ~50-500 rows

↓

Destination entities are created (nodes in this example)

↓

Continues until dataset is fully migrated

Your drush migrate-import simply triggers the Migrate API + queue system.

Commands and internal flow

drush migrate-import upgrade_d7_node

→ Starts migration for nodes.

drush migrate-import upgrade_d7_node --limit=500

→ Migrates 500 records only in this batch.

drush migrate-status

→ Lists available migration plugins + current state.

drush migrate-rollback upgrade_d7_node

→ Rolls back imported content for upgrade_d7_node.

Queue system is fully handled by Drupal. You don't manually manage queues.

Why use --limit=500?

Best practice for large sites:

- Avoids memory/time limit errors
- Forces safe smaller batches

Example batch:

Batch 1 → items 1-500

Batch 2 → items 501-1000

11. Example Custom Data Manipulations in Migrate

Scenario: Modify data on-the-fly using Process Plugins.
Happens during migrate-import for every row.

1. String Transformation

Custom Process Plugin (uppercase + add suffix):

```
namespace Drupal\mymodule\Plugin\migrate\process;
```

```
use Drupal\migrate\Plugin\migrate\process\ProcessPluginBase;
use Drupal\migrate\MigrateExecutableInterface;
use Drupal\migrate\Row;
```

```
/**
 * @MigrateProcessPlugin(
 *   id = "custom_title_modifier"
 * )
 */
class CustomTitleModifier extends ProcessPluginBase {
  public function transform($value, MigrateExecutableInterface $migrate_executable, Row $row, $destination_property) {
    return strtoupper($value) . ' - MIGRATED';
  }
}
```

Input: "summer event 2024"

Output: "SUMMER EVENT 2024 - MIGRATED"

2. Data Cleanup

Custom Process Plugin to clean characters:

```
public function transform($value, MigrateExecutableInterface $migrate_executable, Row $row, $destination_property) {
  return preg_replace('/[^A-Za-z0-9 ]/', '', $value);
}
```

Input: "Grand Opening *** Event!!!"

Output: "Grand Opening Event"

3. Value Mapping

Example mapping array:

```
$mapping = [
  'old_category_code_A' => 'News',
  'old_category_code_B' => 'Blog',
  'old_category_code_C' => 'Press Release',
];
return $mapping[$value] ?? $value;
```

Input: "old_category_code_A"

Output: "News"

4. External API Lookup During Migrate

Example using Google Maps API:

```
$response = \Drupal::httpClient()->get('https://maps.googleapis.com/maps/api/geocode/json', [
  'query' => ['address' => $value, 'key' => 'YOUR_API_KEY']
]);
$data = json_decode($response->getBody(), TRUE);
$lat = $data['results'][0]['geometry']['location']['lat'];
$lng = $data['results'][0]['geometry']['location']['lng'];
$row->setDestinationProperty('field_latitude', $lat);
$row->setDestinationProperty('field_longitude', $lng);
return $value;
```

Input: "1600 Amphitheatre Parkway, Mountain View, CA"

Output: field_latitude=37.422, field_longitude=-122.084

Summary

12. When is Code Required? + Example Custom Plugins

Scenario ■ Code needed? ■ Typical technique
Standard D7 → D10 site ■■ No ■ drush migrate
Custom entities or fields ■■ Yes ■ Custom Destination Plugin
External data sources (CSV, API) ■■ Yes ■ Custom Source Plugin
Overriding batch size globally ■■ (rare) ■ Custom Migrate Executable
Migrating huge standard data with --limit ■■ No ■ Use --limit flag in drush

Conclusion (exact consultant advice)

If it's just a large Drupal 7 site → Drupal 10:

■ No code. Just use drush migrate + batching.

If you are building migrations for:

- External systems
 - Custom entity types
 - Custom data manipulations
- You will write custom Migrate plugins inside a custom module (mymodule).

Example 1: External Systems (CSV / API / external DB)

```
namespace Drupal\mymodule\Plugin\migrate\source;

use Drupal\migrate\Plugin\migrate\source\SqlBase;

/**
 * Migrate external event records into Drupal.
 *
 * @MigrateSource(
 *   id = "external_event"
 * )
 */
class ExternalEvent extends SqlBase {
  public function query() {
    return $this->select('external_events_table', 'e')
      ->fields('e', ['event_id', 'title', 'location', 'start_date']);
  }

  public function fields() {
    return [
      'event_id' => $this->t('Event ID'),
      'title' => $this->t('Title'),
      'location' => $this->t('Location'),
      'start_date' => $this->t('Start date'),
    ];
  }
}
```

```

    public function getIds() {
        return ['event_id' => ['type' => 'integer']];
    }
}

```

Example 2: Custom Entity Types

```

id: custom_event_migration
label: Migrate external events
source:
  plugin: external_event
process:
  title: title
  field_location: location
  field_start_date: start_date
destination:
  plugin: entity:my_custom_event

```

OR any custom entity type:

```

destination:
  plugin: entity:my_custom_entity_type

```

Example 3: Custom Data Manipulation

```

namespace Drupal\mymodule\Plugin\migrate\process;

```

```

use Drupal\migrate\Plugin\migrate\process\ProcessPluginBase;
use Drupal\migrate\MigrateExecutableInterface;
use Drupal\migrate\Row;

```

```

/**
 * Example custom data modifier.
 *
 * @MigrateProcessPlugin(
 *   id = "custom_title_modifier"
 * )
 */

```

```

class CustomTitleModifier extends ProcessPluginBase {
    public function transform($value, MigrateExecutableInterface $migrate_executable, Row $row, $current_step) {
        return strtoupper($value) . ' - MIGRATED';
    }
}

```

Migration YAML:

```

process:
  title:
    plugin: custom_title_modifier

```

source: title

12. When Do You Need Custom Code? + Real-World Examples

In 90% of Drupal 7 → 10 migrations, NO custom code is needed.
You simply use drush + migrate modules.

You only write custom code when:

- You have external data (external DB, CSV, APIs)
- You migrate to a custom entity type
- You want to clean, modify, or map data

Real-World Use Case Summary

Scenario	Code Needed?	Technique
Standard D7 → D10 migration	■ No	drush migrate
External DB, CSV, API data	■ Yes	Custom Source Plugin
Migrate to custom entity types	■ Yes	destination: entity:custom_type
Data cleanup or transformation	■ Yes	Custom Process Plugin

Example 1: External System Data Migration

Goal: Import events from external SQL database.

Plugin file: mymodule/src/Plugin/migrate/source/ExternalEvent.php

```
namespace Drupal\mymodule\Plugin\migrate\source;
use Drupal\migrate\Plugin\migrate\source\SqlBase;

/**
 * @MigrateSource(id = "external_event")
 */
class ExternalEvent extends SqlBase {
  public function query() {
    return $this->select('external_events_table', 'e')
      ->fields('e', ['event_id', 'title', 'location', 'start_date']);
  }
  public function fields() {
    return ['event_id' => $this->t('Event ID'), 'title' => $this->t('Title'),
      'location' => $this->t('Location'), 'start_date' => $this->t('Date')];
  }
  public function getIds() {
    return ['event_id' => ['type' => 'integer']];
  }
}
```

Example 2: Migrate to Custom Entity Type

Migration file example (YAML):

```
id: custom_event_migration
source:
  plugin: external_event
process:
  title: title
  field_location: location
  field_start_date: start_date
destination:
  plugin: entity:my_custom_event
```

Example 3: Data Cleanup / Transformation

Plugin file: mymodule/src/Plugin/migrate/process/CustomTitleModifier.php

```
namespace Drupal\mymodule\Plugin\migrate\process;
use Drupal\migrate\Plugin\migrate\process\ProcessPluginBase;
use Drupal\migrate\MigrateExecutableInterface;
use Drupal\migrate\Row;

/**
 * @MigrateProcessPlugin(id = "custom_title_modifier")
 */
class CustomTitleModifier extends ProcessPluginBase {
  public function transform($value, MigrateExecutableInterface $migrate_executable, Row $row, $current_step) {
    return strtoupper($value) . ' - MIGRATED';
  }
}
```

Usage in migration YAML:

```
process:
  title:
    plugin: custom_title_modifier
    source: title
```

Conclusion

- Use drush migrate for standard sites.
- Write custom Source Plugins to pull external data.
- Write custom Process Plugins to clean, modify, or enrich data.
- Drupal Migrate API handles almost everything. You extend only when needed.

13. Process Plugins vs Large Data Sets

Do Process Plugins help large data migrations?

No, Process Plugins do NOT control batch size or queue handling.

They simply modify or clean a single source row before saving.

Batching & scaling for large sites is always handled by:

```
drush migrate-import upgrade_d7_node --limit=500
```

When do Process Plugins help large migrations?

They improve per-row performance:

- Clean large messy text fields
- Map thousands of legacy values
- Enrich data with external API calls

This reduces memory & processing time for each record.

Example: Heavy Data Cleanup (Custom Process Plugin)

```
namespace Drupal\mymodule\Plugin\migrate\process;  
use Drupal\migrate\Plugin\migrate\process\ProcessPluginBase;  
use Drupal\migrate\MigrateExecutableInterface;  
use Drupal\migrate\Row;
```

```
/**
```

```
 * @MigrateProcessPlugin(  
 *   id = "clean_body_text"  
 * )  
 */
```

```
class CleanBodyText extends ProcessPluginBase {  
  public function transform($value, MigrateExecutableInterface $migrate_executable, Row $row, $source)  
  {  
    $clean = preg_replace('/[^A-Za-z0-9 .,!?\n\r]/', '', $value);  
    return trim($clean);  
  }  
}
```

YAML usage:

```
process:  
  body/value:  
    plugin: clean_body_text  
    source: body
```

Summary: Consultant Best Practice

Option	Purpose

drush migrate-import --limit=500	Safely batch large datasets
Custom Process Plugin	Clean or enrich single row data
Custom Queue Worker (NOT advised)	Core handles migration queues