Connect-CMS V1.41.1 → V2.44.1 アップデート手順書

改訂日：2026年10月23日（実作業の結果を反映）
対象：Gitで設置されたConnect-CMS V1.41.1をV2.44.1へ更新するサイト
想定環境：さくらのレンタルサーバ（FreeBSD）、MySQL 8.0、PHP 8.2、SSH利用可能

> **重要：本書は上記環境での更新成功例に基づく手順です。** 他の環境や独自改修があるサイトにそのまま適用できるとは限りません。作業前に本手順書を生成AIに読み込ませ、対象サイトの環境と更新目的を伝えてください。表示が例と異なる、エラーが出る、判断に迷う場合は**その場で中断**し、コマンドと出力を提示して相談してください。生成AIの判断も誤ることがあるため、不明点を残したまま進めないでください。
>
> パスワード、秘密鍵、`.env` の全文、個人情報を生成AIへ送信しないでください。コマンドの正常出力と確認基準は本書にも記載しています。

1. 事前確認

1. サイト管理者に作業日時と一時停止を周知する。
2. 対象サイトが **Connect-CMS V1.41.1** であることを確認する。
3. サーバーのコントロールパネルでWeb側のPHPが **8.2系** であることを確認する。SSHで次も実行する。

   ```
   php -v
   ```

   目的・正常時： SSHで使われるPHPのバージョンを表示する。例：`PHP 8.2.35 (cli)`。Web側とSSH側の両方が想定のPHP 8.2系であることを確認する。
4. SSH接続後、Connect-CMSの設置先へ移動する。以降のコマンドは、`artisan`、`composer.json`、`.env` がある**プロジェクトのルートディレクトリ**で実行する。

   ```
   cd /home/your-username/www/your-site
   pwd
   ls -a
   ```

   `pwd` で設置先を、`ls -a` で `artisan`、`composer.json`、`.env` の存在を確認する。パスは実際の設置先に読み替える。
5. Composerが利用できることを確認する。プロジェクトルートに `composer.phar` がある場合：

   ```
   php -d allow_url_fopen=1 composer.phar --version
   ```

正常時： Composerのバージョン（例：`Composer version 2.x.x`）が表示される。`composer.phar` がない場合、グローバルにComposerが導入されていれば `composer --version` で確認する。どちらも使用できない場合は中断する。

6. Gitで管理されているプログラムの状態を確認する。

```
git status --short
git describe --tags --exact-match
```

目的・正常時： `git status --short` は変更・未追跡ファイルの一覧を表示し、変更がなければ**何も表示しない**。`git describe --tags --exact-match` は現在のバージョンのタグを表示し、今回は `v1.41.1` が期待値。意図しない変更があれば原因を確認するまで進まない。

> PHP 8.2では旧V1系の一部機能が正常に動かない場合があります。バックアップを確保し、計画したメンテナンス時間内に更新してください。

2. バックアップ

プログラムを更新する前に、以下をWeb公開領域外（作業用PCなど）へ保存します。 データベースのバックアップはphpMyAdminから行えます。

対象	保存元・方法	確認事項
データベース	phpMyAdminの「エクスポート」でSQLファイルを取得	正しいDBか、ファイルが空でないか
デザインテーマ	`public/themes/`	`public/themes/Users/` 以下の独自テーマも含むか
アップロードファイル	`storage/app/uploads/`	フォルダ構造ごと取得できたか
環境設定	`.env`	厳重に保管。必要に応じローカルで `_env` に改名
favicon	`public/uploads/favicon/`	独自faviconを使用している場合のみ取得
robots.txt・サイトマップ	`public/robots.txt`、`public/sitemap.xml`	独自に設置・編集している場合に取得

- FTP/SFTPクライアント（例：Transmit）でファイルを取得してよい。
- サイト独自のプログラム変更やGit管理外の追加ファイルがある場合は、それらも別途保存する。
- ダウンロード後、ファイルとフォルダの存在・サイズを確認する。
- **バックアップが不完全な場合は更新を始めない。** 復元手順も事前に確認しておく。

3. メンテナンスモードにする

```
php artisan down
```

目的・正常時： サイトを一時停止する。例：`Application is now in maintenance mode.` と表示される。ブラウザでもメンテナンス画面が表示されることを確認する。

続いて、**更新前のログの状態を記録**する。

```
ls -lt storage/logs/ | head -10
```

目的・正常時： 更新日時の新しい順にログファイルを表示する。例：`laravel-2026-10-21.log` のような日付付きファイルが表示される場合がある。最新ファイル名と更新日時を記録する。エラーが気になる場合は、実際に存在するファイル名で内容を確認する。

```
# ファイル名は実際のものに置き換える
tail -n 80 storage/logs/laravel-2026-10-21.log
```

目的・正常時： 指定したログの末尾80行を表示する。ログには過去のエラーが残ることもあるため、発生日時を記録して更新後のログと比較する。ログの内容に機密情報がある場合は伏せてから生成AIへ提示する。

> 以降、第11章まで `php artisan up` は実行しない。

4. Connect-CMSのプログラムをV2.44.1に更新する

まず変更状況を確認する。

```
git status --short
```

目的・正常時： メンテナンスモード開始後は `?? storage/framework/maintenance.php` と表示されることがある。これはメンテナンスモードに伴う未追跡ファイルで、**それ自体は異常ではない。削除しない。**その他の予期しない変更がある場合は中断する。

Git管理対象ファイルを独自に変更している場合（例：`public/robots.txt`）： 変更内容を確認し、必ず退避する。

```
git diff -- public/robots.txt
```

退避した内容を確認したうえで、タグ切り替えを妨げる変更があれば元のGit管理状態に戻す。

```
git restore -- public/robots.txt
```

`git restore` はそのファイルの未コミット変更を破棄するため、**バックアップ前に実行しない。** タグ切り替え後に必要な独自設定を復元し、`git diff -- public/robots.txt` で確認する。

Git管理外ファイルがある場合（例：`public/sitemap.xml`）： 対象タグに同名の管理対象ファイルが含まれるかを確認する。

```
git ls-tree -r --name-only tags/v2.44.1 -- public/sitemap.xml
```

何も表示されなければ、対象タグでは管理されていない。退避したうえで、切り替え後にもファイルが残っていることを確認する。同名ファイルが表示される場合は上書き・競合の可能性があるため中断して対処を検討する。

```
git fetch --tags
```

目的・正常時： Gitのリモートリポジトリからバージョンタグを取得する。取得するタグがなければ出力がない場合もある。`fatal` や `error` がなくプロンプトに戻れば次へ進める。

```
git checkout tags/v2.44.1
```

目的・正常時： プログラムをV2.44.1に切り替える。出力例：

```
Previous HEAD position was 04ffb2bfe update: update version to 1.41.1
HEAD is now at 24bdd521a Merge pull request #2476 from opensource-
workshop/v2.44.1
```

コミット番号や説明文は異なることがある。`detached HEAD` の説明が表示されても、タグを直接チェックアウトした場合は通常の動作である。

切り替え後は、次の2つを別々に確認する。

```
git describe --tags --exact-match
```

目的・正常時： `v2.44.1` と表示される。

```
git status --short
```

目的・正常時： 何も表示されないか、メンテナンス中の `?? storage/framework/maintenance.php`、事前に確認した独自ファイルのみが表示される。独自設定を復元した場合はその変更も表示され得る。想定外の変更・エラーがあれば中断する。

5. Composerの依存ライブラリを更新する

```
php -d allow_url_fopen=1 composer.phar install --no-dev --no-interaction
```

目的： V2.44.1が必要とするPHPライブラリを、指定された構成に従ってインストールする。Composerをグローバルに導入している環境では実行方法が異なる場合がある。

正常終了時の出力例（抜粋）：

```
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
   INFO  Discovering packages.
  intervention/image ........ DONE
  laravel/sanctum ........ DONE
  laravel/tinker ........ DONE
58 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
```

成功の判断： パッケージ検出などの処理が `DONE` となり、致命的なエラーなくプロンプトに戻る。パッケージ名・件数は異なることがある。`Package ... is abandoned` は保守上の警告、`funding` は案内であり、それだけでインストール失敗とは判断しない。`Fatal error`、依存関係の解決失敗などがあれば中断する。

> `composer update` に置き換えない。指定バージョンの依存構成を使用するため、`install` を実行する。

6. `.env` の設定を確認する

既存の `.env` を編集する。**新しい `.env` で上書きしない。** エディタの操作方法はここでは扱わない。

メール送信設定が旧形式のときは、設定名を次のように変更する。

変更前：

```
MAIL_DRIVER=smtp
```

変更後：

```
MAIL_MAILER=smtp
```

`smtp` は例。実際に利用しているメール方式の値は維持する。DB接続情報、暗号化キー、メール認証情報など、ほかの設定を不用意に変更しない。

HTTPSで運用する場合のCookie設定

標準の `.env` に次のようなコメントがある場合：

```
# Cookie Secure setting. Set to true for https.
#SESSION_SECURE_COOKIE=true
```

HTTPSで運用するサイトでは、2行目の先頭の `#` を削除して有効にする。

```
# Cookie Secure setting. Set to true for https.
SESSION_SECURE_COOKIE=true
```

この設定により、セッションCookieに `Secure` 属性が付く。すでに同じ設定が有効なら重複して追記しない。

修正後は、設定名と値を確認する。

```
grep -E '^(MAIL_DRIVER|MAIL_MAILER|SESSION_SECURE_COOKIE)=' .env
```

正常時（SMTP・HTTPSサイトの例）： `MAIL_MAILER=smtp` と `SESSION_SECURE_COOKIE=true` が表示され、旧設定 `MAIL_DRIVER` が表示されない。

> `.env` の全文や `MAIL_PASSWORD` の値を生成AIに送信しない。メール送信機能は第12章で確認する。

7. Laravelのキャッシュをクリアする

設定・アプリケーション・画面表示のキャッシュを順に消去する。

```
php artisan config:clear
```

正常時の例： `INFO Configuration cache cleared successfully.`

```
php artisan cache:clear
```

正常時の例： `INFO Application cache cleared successfully.`

```
php artisan view:clear
```

正常時の例： `INFO Compiled views cleared successfully.`

表示文言はLaravelのバージョンで多少異なる。各コマンドがエラーなくプロンプトに戻ることを確認し、異常があれば中断する。

8. データベースのマイグレーション

マイグレーションは、バージョンアップに必要な**データベースの構造・内容の変更処理**である。まず適用状況を表示する。

```
php artisan migrate:status
```

正常時の表示例（抜粋）：

```
Migration name                                                  Batch / Status
2025_12_16_000000_add_depth_to_pages_table                      [5] Ran
2026_03_16_164135_add_select_page_from_searchs                  Pending
```

- `Ran`：既に実行済み。`[5]` などの数字は実行のまとまり（バッチ番号）。
- `Pending`：プログラム側に変更処理は存在するが、現在のデータベースではまだ実行されていない。**更新前に表示されること自体は異常ではない。**
- 未適用件数は、サイトの過去の更新履歴・適用状況によって異なる。今回の更新実績では6件だったが、必ず6件になるわけではない。

SQLバックアップが取得できていることを確認してから、未適用の変更を実行する。

```
php artisan migrate --force
```

正常時の出力例（抜粋）：

```
INFO  Running migrations.
2019_12_14_000001_create_personal_access_tokens_table ..... 56ms DONE
2026_03_16_164135_add_select_page_from_searchs ................ 42ms DONE
```

`--force` は本番環境での実行を許可する指定。**出力全体を生成AIに提示し、実行された処理がすべて `DONE` であるか評価してもらう。** エラーがあれば再実行せず中断する。

完了後、もう一度適用状況を確認する。

```
php artisan migrate:status | grep Pending
```

正常時： 何も表示されなければ未適用のMigrationはない。`Pending` が表示された場合は中断し、原因を確認する。なお、このコマンドは該当行の有無だけを調べるため、実行結果に異常が疑われる場合は `php`

`artisan migrate:status` の全体を確認する。

9. 初期データ（Seeder）の更新

Seeder（シーダー）は、Connect-CMSの動作に必要な初期データや設定情報をデータベースに登録・更新する仕組みである。今回の更新では、V2.44.1に付属する標準Seederを実行する。

標準構成であれば、通常は利用者がSeederのソースコードを一つひとつ調査する必要はない。ただし、**独自にSeederを変更・追加しているサイトでは、実行前に変更内容と影響を確認する。**

```
php artisan db:seed --force
```

正常時の出力例（抜粋）：

```
INFO  Seeding database.
Database\Seeders\DefaultPagesTableSeeder ................... RUNNING
Database\Seeders\DefaultPagesTableSeeder ................... 33 ms DONE
Database\Seeders\DefaultUsersTableSeeder ................... RUNNING
Database\Seeders\DefaultUsersTableSeeder ..................... 9 ms DONE
```

成功の判断： 実行対象のSeederがすべて DONE となり、エラーなくプロンプトに戻る。今回の標準構成での実績は8件だが、表示件数や所要時間は異なる場合がある。**出力全体を生成AIに提示して結果を確認してもらう。** エラーがあれば繰り返し実行せず中断する。

10. 公開前のログ確認

更新前に記録したLaravelログの状態と比較する。

```
ls -lt storage/logs/ | head -6
```

目的・正常時： 新しい順にログファイル名・更新日時を表示する。更新前と同じ日時であれば、少なくとも一覧上は新しいログの生成・更新が見られない。新しいログがあれば、実際のファイル名で内容を確認する。

```
# ファイル名は実際に存在するものに置き換える
tail -n 80 storage/logs/laravel-2026-10-23.log
```

目的・正常時： ログの末尾80行を表示する。ログがない場合は、存在しないファイルを指定して実行しない。過去から残っているエラーと更新中に新しく発生したエラーを区別する。末尾がスタックトレースのみの場合、エラーの冒頭を次のように検索する。

```
# 日付は対象のログファイル名に読み替える
grep -n -E 'production.ERROR|local.ERROR' storage/logs/laravel-2026-10-23.log | tail -10
```

ログ一覧と必要に応じて確認したログの内容を生成AIに提示し、更新前と比較して評価してもらう。 機密情報は伏せる。新たな重大エラーがある場合は原因を確認するまで解除しない。

11. メンテナンスモードを解除する

```
php artisan up
```

目的・正常時： サイトの公開を再開する。例：`INFO Application is now live.` と表示される。

ブラウザで `https://www.example.com` を再読み込みし、トップページが表示されることを確認する。

500エラーなどの重大な不具合が出た場合は、必要に応じて直ちに次のコマンドでメンテナンスモードに戻し、ログを調査する。

```
php artisan down
```

正常時： `Application is now in maintenance mode.` に相当するメッセージが表示される。

12. 動作確認

次の項目を確認する。閲覧テストはログアウト状態（シークレットウィンドウなど）でも行う。

- ☐ トップページが正常に表示され、画像・メニュー・フッターが崩れていない
- ☐ 管理者ログイン画面が表示され、既存アカウントでログインできる
- ☐ 管理画面で **バージョン2.44.1** を確認できる
- ☐ 「ページ管理」に既存ページと階層が表示される
- ☐ 主要ページの本文、画像、ブログ、カレンダーなどが表示される
- ☐ アップロード済みPDFなどを開け、内容を読める
- ☐ 独自テーマを使用している場合、配色・レイアウトを確認できる
- ☐ テスト可能な記事で編集画面の表示・一時的な追記・保存・再表示ができる
- ☐ テストの追記を削除して再保存し、元の状態に戻せる
- ☐ 「システム管理 → メール送信テスト」でテストメールを送信し、指定した宛先で受信できる
- ☐ お問い合わせフォームなどを実際に送信し、送信完了画面、管理者への通知メール、問い合わせ者への自動返信メールを確認できる
- ☐ 実運用に必要な検索、ファイルダウンロードなどを確認できる

既存データを破壊しない方法でテストする。 実施できなかった機能は「未確認」として記録する。

メール送信時にHTTP 500エラーなどが出た場合： Laravelログで原因を確認する。SMTP認証エラーなら、`.env` の `MAIL_MAILER`、`MAIL_HOST`、`MAIL_PORT`、`MAIL_USERNAME`、`MAIL_ENCRYPTION` とメールサービス側の認証設定を確認する。`MAIL_PASSWORD` の値は画面共有・生成AIへの提示を避ける。Google

Workspace等でアプリパスワードを利用している場合は、当該アカウントに有効なアプリパスワードがあるか確認し、必要に応じて再発行する。設定変更後は `php artisan config:clear` を実行し、管理画面のメール送信テストと実フォームの送信を再確認する。

過去のエラーと区別する場合： ログに記録された日時を確認し、更新前から同種のエラーが発生していたか調べる。メール認証エラーは必ずしもアップデートに起因しない。

13. Laravelログの最終確認

```
ls -lt storage/logs/ | head -6
```

目的・正常時： 第3章・第10章で確認したログと比較し、動作確認中に新しいログが発生したか確認する。新規ログがなければ、その旨を記録する。

新しいログがある場合は、実際に存在するファイル名で末尾を確認する。

```
# ファイル名は実際に存在するものに置き換える
tail -n 80 storage/logs/laravel-2026-10-23.log
```

目的・正常時： 新しい重大なエラーがないか確認する。ログがない場合は実行不要。必要なら、機密情報を伏せた出力を生成AIに提示して判断を補助してもらう。

> 新規ログがないことは良い兆候だが、すべての機能に問題がないことの証明ではない。第12章の動作確認と合わせて判断する。

14. 作業完了後

1. SQL、テーマ、アップロードファイル、`.env`、favicon（使用時）のバックアップを安全な場所に保管する。
2. 作業日時、更新前後のバージョン、Migration・Seederの結果、動作確認結果、未確認機能を記録する。
3. サイト管理者や顧客にメンテナンス終了を連絡する。
4. 数日間はサイト表示とLaravelログを適宜確認する。運用開始後も、お問い合わせフォームからの送信・通知・自動返信を定期的にテストする。
5. 不具合が発生した場合、**旧バージョンのプログラムだけを戻して公開しない。** データベースも更新されているため、復元はバックアップとの整合性を確保して行う。

参考資料

- Connect-CMS公式Wiki：Update（アップデート）
- Connect-CMS GitHub

作業記録欄（記入例）

項目	記録
対象サイト	`https://www.example.com`
作業日	YYYY-MM-DD
更新前	V1.41.1
更新後	V2.44.1
PHP（Web / CLI）	8.2.x / 8.2.x
DBバックアップ	取得済み／未取得
ファイルバックアップ	取得済み／未取得
Migration	正常／要確認
Seeder	正常／要確認
動作確認	正常／一部未確認／要対応
最終ログ	新規重大エラーなし／要確認
メンテナンス終了連絡	済／未

備考