Migration Repair Doctor v1.5

Installation, audit et réparation des migrations Django

Guide opérationnel complet - Django + MySQL/MariaDB

Commande : python manage.py migration_repair_doctor

/static/toolbox/migrationrepair_doctor.py

IDEO-Lab - Mai 2026

Table des matières

1. Objectif de l'addon et problème traité
2. Installation complète dans un projet Django
3. Commandes rapides à retenir
4. Mode central --repair-partial
5. Référence CLI des options importantes
6. Cas d'usage et séquences exactes
7. Cron, CI et exploitation
8. Sécurité, backups et bonnes pratiques
9. Dépannage et lecture des sorties

Document généré
Ce PDF reprend l'installation et l'utilisation de l'addon Migration Repair Doctor v1.5. Il est conçu comme un runbook: installation, commandes, scénarios, dépannage et exploitation.

1. Objectif de l'addon et problème traité

Migration Repair Doctor v1.5 est une commande Django qui inspecte et répare les désynchronisations entre models.py, les fichiers de migration, la table django_migrations et le schéma réel MySQL/MariaDB.

Le cas le plus critique est la migration partiellement appliquée: une table a été créée, certaines colonnes existent, un index échoue, mais Django n'a pas enregistré la migration comme appliquée. Au run suivant, Django tente de rejouer tout le fichier et l'incident se répète.

> Cas typique
> Erreur MySQL 1071: clé trop longue sur un index. La table a été créée, mais l'index final échoue. La migration reste non appliquée et Django retente ensuite de créer des objets déjà présents.

Source d'état	Exemple	Ce que le doctor vérifie
models.py	TechHtmlKeywordMatch	Tables, champs, ForeignKey, max_length, null, index déclarés côté modèle.
Fichiers migrations	0014_xxx.py	Opérations CreateModel, AddField, AlterField, AddIndex.
django_migrations	app/name/applied	Historique réellement déclaré comme appliqué par Django.
information_schema	COLUMNS / STATISTICS	Schéma réel MariaDB: tables, colonnes, index, FK.

Ce que le script sait faire

- Scanner les migrations d'une app et scorer les candidates cassées.
- Vérifier l'état réel DB vs models.py et lister les colonnes, tables ou index manquants.
- Réécrire une migration cible avec uniquement les opérations déjà matérialisées en base.
- Fake-record une migration nettoyée dans django_migrations.
- Générer une migration follow-up contenant les champs ou corrections encore manquants.
- Appliquer migrate puis relancer une vérification finale.
- Exporter un rapport Excel ou PDF pour audit.

2. Installation complète dans un projet Django

Le fichier public de téléchargement est disponible à l'URL suivante:

Download public

```
/static/toolbox/migrationrepair_doctor.py
```

> Nom du fichier côté Django
> Pour obtenir la commande python manage.py migration_repair_doctor, le fichier doit être installé dans management/commands/ sous le nom migration_repair_doctor.py. Le nom public téléchargé peut être différent.

2.1 Arborescence attendue

Arborescence Django

```
your_app/
    management/
        __init__.py
        commands/
            __init__.py
            migration_repair_doctor.py
```

2.2 Dépendances Python

Le script utilise Django, openpyxl pour l'export Excel et reportlab pour l'export PDF.

Installation dépendances

```
pip install openpyxl reportlab
```

2.3 Vérifier que la commande est visible

Vérification

```
python manage.py help migration_repair_doctor
python manage.py migration_repair_doctor --help
```

2.4 Premier run sans risque

Par défaut, si ni --execute ni --simulation ne sont fournis, le script bascule en simulation. Pour un premier diagnostic, utilisez --verify.

Premier test safe

```
python manage.py migration_repair_doctor   --app tech_glossary   --verify   --verbose
```

2.5 Checklist avant production

- Faire un backup Git ou au minimum sauvegarder le dossier migrations/.

- Faire un backup SQL ou un snapshot de base si l'environnement est en production.
- Vérifier que l'utilisateur DB peut lire information_schema et écrire dans django_migrations.
- Commencer par --simulation pour toute réparation partielle.
- Ne pas relancer migrate en boucle sur la même migration cassée.

3. Commandes rapides à retenir

> Commande principale
> Pour une migration partiellement appliquée, la commande à retenir est --repair-partial. Elle remplace la longue séquence manuelle clean + fake + verify + migrate + verify.

3.1 Vérifier une app

Verify read-only

```
python manage.py migration_repair_doctor   --app tech_glossary   --verify   --verbose
```

3.2 Générer une migration follow-up depuis verify

Verify + follow-up

```
python manage.py migration_repair_doctor   --app tech_glossary   --verify   --execute
  --verbose

python manage.py migrate tech_glossary

python manage.py migration_repair_doctor   --app tech_glossary   --verify   --verbose
```

3.3 Réparer une migration partiellement appliquée en simulation

Repair partial simulation

```
python manage.py migration_repair_doctor   --app tech_glossary   --repair-partial
  --simulation   --verbose
```

3.4 Réparer réellement une migration partiellement appliquée

Repair partial execute

```
python manage.py migration_repair_doctor   --app tech_glossary   --repair-partial
  --execute   --verbose
```

3.5 Réparer une migration précise

Repair partial target

```
python manage.py migration_repair_doctor   --app tech_glossary   --target 0014
  --repair-partial   --execute   --verbose
```

3.6 Exporter un rapport Excel

Export report

```
python manage.py migration_repair_doctor   --app tech_glossary   --scan   --export
  excel   --open-export   --verbose
```

4. Mode central --repair-partial

Le mode --repair-partial est le runbook automatique pour les migrations interrompues. Il évite d'avoir à enchaîner manuellement quatre commandes et limite les erreurs humaines.

> Important
> Ne lancez pas migrate brut en boucle. Lorsqu'une migration est suspectée partiellement appliquée, lancez d'abord --repair-partial --simulation --verbose.

Étape	Action	Résultat attendu
1	Scan ou target	Le script choisit la migration cassée avec --scan ou utilise --target.
2	Clean target migration	Réécriture de la migration avec les opérations réellement présentes en DB.
3	Fake-record	Insertion contrôlée dans django_migrations si pas déjà appliquée.
4	Follow-up sync	Génération d'une migration complémentaire pour les champs/colonnes manquants.
5	Migrate + verify	Application de la follow-up puis vérification finale Status: OK.

Commande standard

--repair-partial standard

```
python manage.py migration_repair_doctor   --app your_app   --repair-partial
  --execute   --verbose
```

Commande ciblée

--repair-partial avec target

```
python manage.py migration_repair_doctor   --app your_app   --target 0050   --repair-
  partial   --execute   --verbose
```

Sortie attendue

Exemple de sortie

```
REPAIR PARTIAL MIGRATION WORKFLOW
App: your_app
Mode: EXECUTE

STEP 1/5 - Clean target migration
Backup created: ...
Rewritten cleaned migration: ...

STEP 2/5 - Fake-record cleaned migration
Recorded migration as applied: your_app.0050_...

STEP 3/5 - Generate follow-up sync migration
Generated migration: 0051_your_app_manual_field_sync.py
Verification failed as expected before follow-up migration is applied.

STEP 4/5 - Run migrate
Applying your_app.0051_your_app_manual_field_sync... OK

STEP 5/5 - Final verification
Errors count      : 0
Pending AddField  : 0
Pending AlterField: 0
Pending AddIndex  : 0
Pending CreateModel: 0
Status: OK
```

5. Référence CLI des options importantes

Option	Type	Description
--app	required	App Django à inspecter ou réparer.
--target	safe	Préfixe ou nom exact de migration, par exemple --target 0014.
--last	safe	Utilise la dernière migration sur disque.
--scan / --explore	safe	Scanne et score les migrations suspectes.
--verify	safe	Vérifie DB vs models.py; avec --execute, génère une migration follow-up.
--clean-target-migration	mutate	Réécrit la migration cible nettoyée selon la DB réelle.
--fake-record	mutate	Enregistre la migration cible dans django_migrations.
--repair-partial	workflow	Workflow complet de réparation partielle en cinq étapes.
--safe-threshold	safe	Seuil de longueur sûre pour index CharField MySQL/MariaDB; défaut: 191.
--export	safe	Exporte un rapport en excel, pdf ou both.
--backup-dir	safe	Dossier de backup des migrations réécrites.
--simulation	safe	Affiche les actions sans écrire disque/DB.
--execute	mutate	Autorise les écritures disque/DB.

Règle pratique
--verify, --scan et --presim sont des modes d'analyse. --execute déclenche les écritures. --repair-partial est le workflow complet.

6. Cas d'usage et séquences exactes

Cas A - Erreur MySQL 1071: clé trop longue

Symptôme: MariaDB/MySQL refuse un index sur un champ long, souvent CharField(260), URLField ou TextField.

Cas A

```
python manage.py migration_repair_doctor   --app your_app   --repair-partial
  --simulation   --verbose

python manage.py migration_repair_doctor   --app your_app   --repair-partial
  --execute   --verbose
```

Après réparation
Corriger models.py: ne pas mettre d'index direct sur des VARCHAR/TEXT longs. Utiliser un hash court ou une stratégie applicative.

Cas B - Tables créées mais migration non appliquée

Cas B

```
python manage.py migration_repair_doctor   --app your_app   --scan   --verbose

python manage.py migration_repair_doctor   --app your_app   --repair-partial
  --execute   --verbose
```

Cas C - Colonnes FK manquantes après fake-record

Cas C

```
python manage.py migration_repair_doctor   --app your_app   --verify   --execute
  --verbose

python manage.py migrate your_app

python manage.py migration_repair_doctor   --app your_app   --verify   --verbose
```

Cas D - Diagnostic seulement, aucun changement

Cas D

```
python manage.py migration_repair_doctor   --app your_app   --scan   --export excel
  --open-export   --verbose
```

Cas E - Migration précise connue

Cas E

```
python manage.py migration_repair_doctor   --app your_app   --target 0050   --clean-
  target-migration   --fake-record   --execute   --verbose

python manage.py migration_repair_doctor   --app your_app   --verify   --execute
  --verbose

python manage.py migrate your_app
```

7. Cron, CI et exploitation

7.1 Cron quotidien de vérification

Cron verify

```
# Tous les jours à 03:20
20 3 * * * cd /var/www/ideo_lab && /opt/venv/bin/python manage.py
  migration_repair_doctor --app tech_glossary --verify --verbose >>
  /var/log/ideo_lab/migration_repair_doctor.log 2>&1
```

7.2 CI avant déploiement

CI gate

```
python manage.py migration_repair_doctor --app tech_glossary --verify --verbose
python manage.py showmigrations tech_glossary
```

7.3 Runbook production en cas d'incident

- Stopper les relances migrate: ne pas rejouer en boucle une migration cassée.
- Faire un backup SQL ou snapshot infra.
- Lancer une simulation: --repair-partial --simulation --verbose.
- Si le plan est cohérent, lancer --repair-partial --execute --verbose.
- Contrôler que --verify --verbose termine avec Status: OK.

8. Sécurité, backups et bonnes pratiques

Principe
Le script modifie potentiellement les fichiers de migration et django_migrations. Il ne remplace pas un backup SQL en production.

Action	Risque	Bonne pratique
--verify	Faible	Read-only sauf si combiné à --execute pour générer une follow-up.
--scan	Faible	Diagnostic et scoring; idéal avant toute réparation.
--clean-target-migration	Moyen	Réécrit un fichier migration; vérifier le backup généré.
--fake-record	Moyen	Modifie django_migrations; à utiliser après nettoyage cohérent.
--repair-partial	Élevé mais encadré	Faire simulation puis execute; contrôler Status: OK.

Règles MySQL/MariaDB sur les index

Champ	Index direct ?	Conseil
CharField <= 191	Oui	OK en utf8mb4.
CharField 220/260/500	Non	Utiliser un hash court ou index applicatif.
TextField	Non	Jamais d'index BTree direct standard.
URLField long	Non	Indexer un hash ou une clé courte.
ForeignKey	Oui	Index automatique généralement créé par Django/MySQL.

9. Dépannage et lecture des sorties

Erreur: Use either --simulation or --execute, not both

Correction

```
# Bon
python manage.py migration_repair_doctor --app your_app --repair-partial --simulation

# Bon
python manage.py migration_repair_doctor --app your_app --repair-partial --execute
```

Erreur: No materialized operations detected

Le script ne voit aucune table ou colonne réellement présente pour construire une migration nettoyée. Soit la mauvaise migration est ciblée, soit la DB n'a rien matérialisé.

Diagnostic

```
python manage.py migration_repair_doctor --app your_app --scan --verbose
python manage.py showmigrations your_app
```

Verify affiche des warnings d'index manquants mais Status: OK

Ce n'est pas bloquant si les erreurs sont à zéro. Les warnings peuvent signaler des index déclarés côté modèle mais absents côté DB ou renommés par MySQL.

Le follow-up est généré mais migrate échoue

Contrôle follow-up

```
python manage.py showmigrations your_app
python manage.py sqlmigrate your_app 0051
python manage.py migrate your_app
```

Status final attendu
Un système réparé doit finir avec: Errors count = 0, Pending AddField = 0, Pending AlterField = 0, Pending AddIndex = 0, Pending CreateModel = 0, Status: OK.