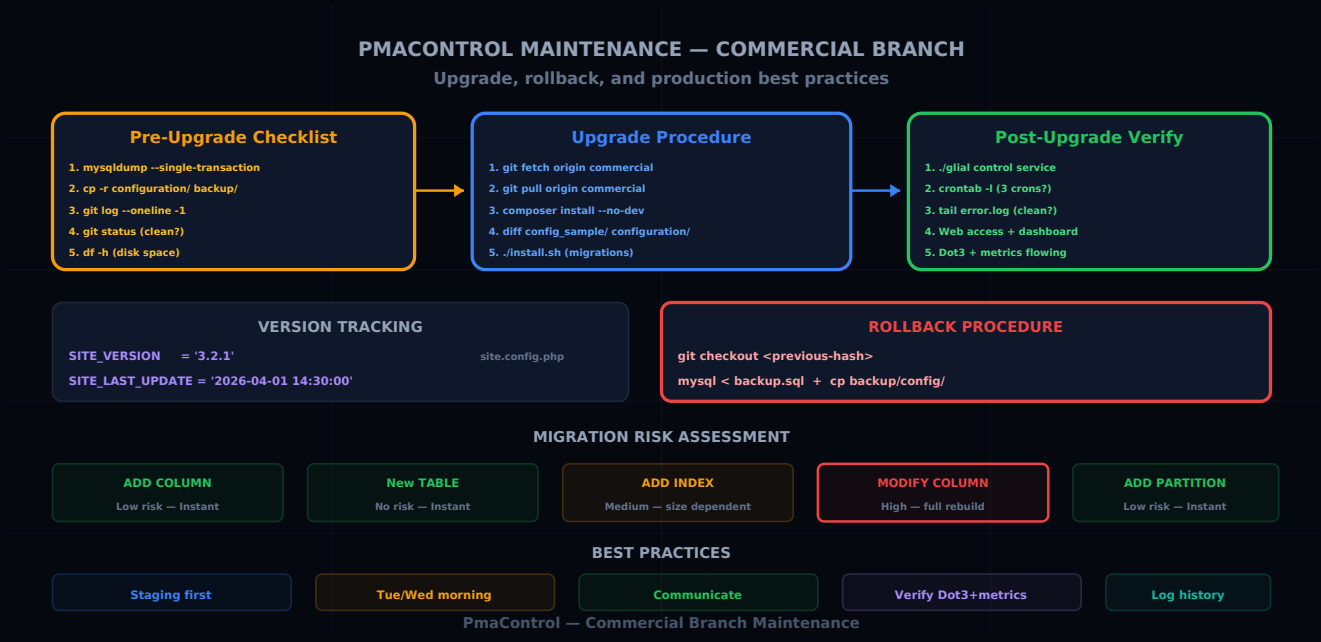
Maintenir PmaControl branche commercial : upgrade, rollback et bonnes pratiques

Aurélien LEQUOY · 13 avril 2026

PMACONTROL UPGRADE MAINTENANCE GIT PRODUCTION



PmaControl est un déploiement Git

PmaControl n'est pas distribué sous forme de paquet .deb ou .rpm. C'est un déploiement basé sur Git : vous clonez le dépôt, vous installez les dépendances Composer, vous exécutez le script d'installation, et c'est en production.

```
git clone -b commercial https://github.com/pmacontrol/pmacontrol.git /srv/www/pmacontrol
cd /srv/www/pmacontrol
composer install --no-dev
./install.sh
```

Ce modèle a des avantages (mise à jour rapide, rollback facile, pas de gestionnaire de paquets) et des inconvénients (pas de gestion automatique des dépendances système, pas de scripts post-install standardisés). Ce guide couvre la maintenance au quotidien.

Suivi de version

PmaControl stocke sa version dans le fichier de configuration du site :

```
// configuration/site.config.php
define('SITE_VERSION', '3.2.1');
define('SITE_LAST_UPDATE', '2026-04-01 14:30:00');
```

Ces constantes sont mises à jour automatiquement lors de l'installation (`install.sh`). Vous pouvez les consulter :

- Via l'interface web : page "À propos" ou footer
- Via le CLI : `grep SITE_VERSION configuration/site.config.php`
- Via l'API : `GET /api/v1/status` retourne la version

Avant toute mise à jour, notez la version courante :

```
cd /srv/www/pmacontrol
grep -E 'SITE_VERSION|SITE_LAST_UPDATE' configuration/site.config.php
```

Checklist pré-upgrade

Avant de lancer la mise à jour, exécutez cette checklist complète :

1. Sauvegarder la base de données

```
mysqldump --single-transaction --routines --triggers \
  -u pmacontrol -p pmacontrol > /backup/pmacontrol_$(date +%Y%m%d_%H%M%S).sql
```

Le flag `--single-transaction` est essentiel : il garantit un backup cohérent sans verrouiller les tables (InnoDB/RocksDB).

Vérifier la taille du dump :

```
ls -lh /backup/pmacontrol_*.sql
```

Un dump vide ou anormalement petit indique un problème.

2. Sauvegarder la configuration

```
cp -r /srv/www/pmacontrol/configuration/ /backup/pmacontrol_config_$(date +%Y%m%d)/
```

Les fichiers de configuration sont les plus critiques : `db.config.ini.php` (credentials), `telegram.php`, `acl.config.ini`, `site.config.php`. Un upgrade ne devrait pas les modifier, mais une erreur humaine est vite arrivée.

3. Noter la version courante et le commit

```
cd /srv/www/pmacontrol
git log --oneline -1
# fe0911d (HEAD -> commercial) Fix: replication display for MySQL 8.4

grep SITE_VERSION configuration/site.config.php
# define('SITE_VERSION', '3.2.1');
```

Conservez le hash du commit (ici `fe0911d`) — c'est votre point de rollback.

4. Vérifier l'état Git

```
git status
```

S'il y a des modifications locales non commitées, décidez maintenant : les commiter, les stasher, ou les abandonner. Un `git pull` avec des modifications locales peut générer des conflits.

```
# Si des modifications doivent être conservées
git stash save "pre-upgrade $(date +%Y%m%d)"

# Si elles doivent être abandonnées
git checkout -- .
```

5. Vérifier l'espace disque

```
df -h /srv/www/pmacontrol/
df -h /var/lib/mysql/
```

L'upgrade et les migrations peuvent nécessiter de l'espace temporaire. Assurez-vous d'avoir au moins 20% d'espace libre sur les deux partitions.

Procédure d'upgrade

Étape 1 : Récupérer les changements

```
cd /srv/www/pmacontrol
git fetch origin commercial
```

Avant de merger, examinez ce qui a changé :

```
git log --oneline HEAD..origin/commercial
```

Cela liste tous les commits entre votre version et la dernière. Lisez les messages de commit pour identifier :

- Des changements de schéma de base de données
- Des modifications de configuration
- Des dépendances Composer ajoutées ou modifiées
- Des breaking changes signalés

Étape 2 : Appliquer la mise à jour

```
git pull origin commercial
```

Si des conflits apparaissent, ils sont presque toujours dans les fichiers de configuration. Ne résolvez jamais un conflit en acceptant aveuglément "theirs" — vérifiez manuellement.

Étape 3 : Mettre à jour les dépendances

```
composer install --no-dev
```

`composer install` (sans `update`) utilise le `composer.lock` du dépôt, ce qui garantit les mêmes versions que l'équipe de développement. N'utilisez **jamais** `composer update` en production — cela pourrait tirer des versions non testées.

Vérifiez que l'installation s'est bien passée :

```
# Vérifier qu'il n'y a pas d'erreur
echo $?
# Doit retourner 0
```

```
# Vérifier le dossier vendor
ls -la vendor/autoload.php
```

Étape 4 : Vérifier les changements de configuration

Comparez les fichiers échantillons avec votre configuration actuelle :

```
diff -r config_sample/ configuration/ --brief
```

Si de nouveaux fichiers apparaissent dans `config_sample/` qui n'existent pas dans `configuration/`, ce sont de nouvelles configurations à intégrer :

```
# Lister les fichiers nouveaux dans config_sample
for f in config_sample/*; do
    base=$(basename "$f")
    if [ ! -f "configuration/$base" ]; then
        echo "NEW: $base — needs to be copied and configured"
    fi
done
```

Pour chaque nouveau fichier :

```
cp config_sample/new_config.php configuration/new_config.php
# Éditer et adapter les valeurs à votre environnement
```

Étape 5 : Exécuter les migrations

```
./install.sh
```

Le script `install.sh` gère les migrations de schéma de base de données. Il :

1. Détecte la version actuelle du schéma
2. Applique les migrations manquantes dans l'ordre
3. Met à jour `SITE_VERSION` et `SITE_LAST_UPDATE`

Vérification manuelle recommandée : avant d'exécuter `install.sh`, examinez les migrations :

```
# Lister les fichiers de migration
ls -la data/migrations/ 2>/dev/null || ls -la install/migrations/ 2>/dev/null
```

Si une migration contient des `ALTER TABLE` sur des tables volumineuses (comme `ts_value_general_int`), prévoyez un temps de maintenance plus long.

Checklist post-upgrade

1. Relancer les services

```
./glial control service
```

Cette commande redémarre le cycle de collecte et de traitement. Vérifiez qu'elle ne retourne pas d'erreur.

2. Vérifier les crons

```
crontab -l -u www-data
```

Vérifiez que les trois crons essentiels sont présents :

```
* * * * * cd /srv/www/pmacontrol && ./glial agent check_daemon >> /tmp/pmacontrol_agent.log 2>&1
* * * * * cd /srv/www/pmacontrol && ./monitor_mysql.sh >> /tmp/pmacontrol_monitor.log 2>&1
0 */4 * * * cd /srv/www/pmacontrol && ./glial control service >> /tmp/pmacontrol_control.log 2>&1
```

3. Vérifier les logs d'erreur

```
# Logs PHP
tail -20 /var/log/apache2/error.log

# Logs PmaControl
tail -20 /tmp/pmacontrol_agent.log
tail -20 /tmp/pmacontrol_monitor.log
tail -20 /tmp/pmacontrol_control.log
```

Cherchez les erreurs fatales, les warnings de classes manquantes, les erreurs de base de données. Un upgrade réussi ne devrait générer aucune erreur nouvelle.

4. Vérifier l'accès web

Ouvrez PmaControl dans le navigateur et vérifiez :

- La page de login fonctionne
- Le dashboard principal se charge
- La liste des serveurs s'affiche
- Un serveur individuel est accessible
- La page slave fonctionne (si applicable)
- La topologie Dot3 se charge

5. Vérifier les métriques

Attendez 5 minutes (un cycle de collecte complet) puis vérifiez :

```
# Les agents collectent-ils ?
./glial agent check_daemon

# Les données arrivent-elles ?
mysql -u pmacontrol -p pmacontrol -e "
SELECT server_id, MAX(timestamp) as last_data
FROM ts_value_general_int
GROUP BY server_id
HAVING last_data < NOW() - INTERVAL 5 MINUTE;
"
```

Si cette requête retourne des résultats, certains serveurs ne reçoivent plus de données — investiguer.

6. Vérifier Dot3 et la topologie

```
./glial dot3 generate
```

Vérifiez que la topologie se régénère sans erreur et que le graphe dans l'interface web est cohérent.

Procédure de rollback

Si quelque chose tourne mal, voici comment revenir en arrière.

Rollback du code

```
cd /srv/www/pmacontrol
git checkout <previous-commit-hash>
composer install --no-dev
```

Par exemple, si le commit pre-upgrade était `fe0911d` :

```
git checkout fe0911d
composer install --no-dev
```

Rollback de la base de données (si le schéma a changé)

Si `install.sh` a modifié le schéma, il faut restaurer le backup :

```
mysql -u root -p pmacontrol < /backup/pmacontrol_20260413_143000.sql
```

Attention : cette opération écrase toutes les données collectées depuis le backup. Si l'upgrade a eu lieu il y a 2 heures, vous perdez 2 heures de métriques. C'est pourquoi l'upgrade doit être planifié pendant une fenêtre de maintenance.

Rollback de la configuration

```
cp /backup/pmacontrol_config_20260413/* /srv/www/pmacontrol/configuration/
```

Après le rollback

```
./glial control service
# Vérifier les logs
tail -20 /tmp/pmacontrol_agent.log
# Vérifier l'accès web
curl -s -o /dev/null -w "%{http_code}" https://pmacontrol.example.com/
# Devrait retourner 200
```

Gestion des dépendances Composer

Comprendre le lock file

Le fichier `composer.lock` est versionné dans le dépôt. Il garantit que tout le monde utilise exactement les mêmes versions de dépendances. Quand `composer.lock` change dans un pull :

```
# Voir les changements de dépendances
git diff HEAD~1 composer.lock | grep '"name"'
```

Vérifier les breaking changes

Si une dépendance majeure change (par exemple, Glial framework de 2.x à 3.x), c'est un changement risqué. Vérifiez le CHANGELOG de la dépendance :

```
# Lister les dépendances mises à jour
composer show --latest --outdated
```

Ne jamais composer update en production

```
# NON — tire les dernières versions possibles
composer update

# OUI — installe exactement les versions du lock file
composer install --no-dev
```

Migrations de base de données

Comment elles fonctionnent

`install.sh` exécute les migrations séquentiellement. Chaque migration est idempotente — elle vérifie d'abord si la modification a déjà été appliquée :

```
-- Exemple de migration typique
-- Vérifie si la colonne existe avant de l'ajouter
SET @exist := (SELECT COUNT(*) FROM information_schema.COLUMNS
               WHERE TABLE_SCHEMA = 'pmacontrol'
               AND TABLE_NAME = 'mysql_server'
               AND COLUMN_NAME = 'new_column');

SET @sql = IF(@exist = 0,
    'ALTER TABLE mysql_server ADD COLUMN new_column VARCHAR(255) DEFAULT NULL',
```

```
    'SELECT "Column already exists"');
PREPARE stmt FROM @sql;
EXECUTE stmt;
```

Migrations risquées

Certaines migrations sont plus risquées que d'autres :

Type	Risque	Temps
ADD COLUMN (nullable)	Faible	Instant (MariaDB 10.0+)
ADD INDEX	Moyen	Proportionnel à la taille
MODIFY COLUMN (type change)	Élevé	Full table rebuild
DROP COLUMN	Faible	Instant (MariaDB 10.4+)
ADD PARTITION	Faible	Instant
Nouvelle table	Aucun	Instant

Pour les tables volumineuses (comme `ts_value_general_int` qui peut faire plusieurs Go), un `ADD INDEX` peut prendre des minutes voire des heures. Planifiez en conséquence.

Revue manuelle recommandée

Avant d'exécuter `install.sh`, lisez les fichiers de migration pour comprendre ce qui va être modifié. Si une migration vous semble risquée (ALTER TABLE sur une table de plusieurs Go), testez-la d'abord sur un environnement de staging.

Bonnes pratiques

1. Environnement de staging

Maintenez un environnement de staging qui réplique votre production. L'upgrade y est testé avant d'être appliqué en production :

```
1. Staging: git pull → composer install → install.sh → vérification
2. Attendre 24h — observer les logs
3. Production: même procédure
```

Le staging n'a pas besoin de superviser autant de serveurs que la production. 5-10 serveurs MariaDB / MySQL suffisent pour valider le bon fonctionnement.

2. Fenêtre de maintenance planifiée

Ne faites jamais un upgrade un vendredi à 17h. Planifiez :

- **Mardi ou mercredi** : en milieu de semaine, l'équipe est disponible
- **Matin** : pour avoir la journée entière pour détecter les problèmes
- **Hors pic** : pas pendant un déploiement applicatif ou un batch nocturne

3. Communiquer

Prévenez l'équipe avant l'upgrade :

```
Objet : Maintenance PmaControl — Mar 13 avril 09:00-10:00

L'instance PmaControl sera mise à jour de la version 3.2.1 à 3.3.0.
Pendant la maintenance (environ 30 minutes) :
- Les dashboards peuvent être temporairement indisponibles
- La collecte de métriques sera interrompue (rattrapage automatique)
- Les alertes Telegram seront suspendues puis reprises

Contact : dba@company.com
```

4. Vérifier Dot3 + métriques après upgrade

C'est le test de fumée le plus important. Si les métriques arrivent et que la topologie est correcte, l'upgrade est réussi. Si l'un des deux ne fonctionne pas, il y a un problème à investiguer.

5. Garder un historique des upgrades

Maintenez un fichier simple qui liste les upgrades effectués :

```
2026-04-13 09:15 | 3.2.1 → 3.3.0 | fe0911d → a1b2c3d | OK
2026-03-15 10:00 | 3.1.0 → 3.2.1 | 1234abc → fe0911d | OK — migration ts_value lente (45min)
2026-02-01 08:30 | 3.0.5 → 3.1.0 | 9876def → 1234abc | ROLLBACK — bug #342 dans Listener
```

6. Automatiser quand c'est mûr

Une fois que vous avez fait 5-10 upgrades manuels sans incident, vous pouvez automatiser avec un script :

```
#!/bin/bash
# upgrade_pmacontrol.sh
set -euo pipefail

BACKUP_DIR="/backup/pmacontrol/$(date +%Y%m%d_%H%M%S)"
mkdir -p "$BACKUP_DIR"

# Backup
mysqldump --single-transaction -u pmacontrol -p"$DB_PASS" pmacontrol > "$BACKUP_DIR/db.sql"
cp -r /srv/www/pmacontrol/configuration/ "$BACKUP_DIR/config/"
git -C /srv/www/pmacontrol log --oneline -1 > "$BACKUP_DIR/version.txt"

# Upgrade
cd /srv/www/pmacontrol
git pull origin commercial
composer install --no-dev
./install.sh

# Verify
./glial control service
curl -s -o /dev/null -w "%{http_code}" https://pmacontrol.example.com/ | grep -q 200

echo "Upgrade successful"
```

Mais gardez toujours la possibilité d'un rollback manuel.

Erreurs courantes

"Class not found" après upgrade

Symptôme : erreur PHP `Class 'Xyz' not found` dans les logs.

Cause : `composer install` n'a pas été exécuté après le pull, ou l'autoloader n'a pas été régénéré.

Solution :

```
composer install --no-dev
composer dump-autoload
```

Erreur de migration "Table already exists"

Symptôme : `install.sh` échoue avec `Table 'xyz' already exists`.

Cause : la migration n'est pas idempotente (bug dans le script de migration).

Solution : vérifier manuellement si la table/colonne existe et sauter la migration si nécessaire. Signaler le bug à l'équipe PmaControl.

Conflits Git dans la configuration

Symptôme : `git pull` échoue avec des conflits dans `configuration/`.

Cause : vous avez modifié un fichier de configuration qui a aussi été modifié upstream.

Solution :

```
# Sauvegarder votre version
cp configuration/problematic_file.php /tmp/

# Accepter la version upstream
git checkout --theirs configuration/problematic_file.php
git add configuration/problematic_file.php

# Ré-appliquer vos modifications manuellement
# Comparer /tmp/problematic_file.php avec la version upstream
```

Le stash est perdu

Symptôme : vous avez fait `git stash` avant l'upgrade et ne retrouvez plus vos modifications.

Solution :

```
git stash list
# stash@{0}: On commercial: pre-upgrade 20260413

git stash pop stash@{0}
```

Conclusion

Maintenir PmaControl en production est un processus prévisible si vous suivez la procédure : backup, pull, composer install, install.sh, vérification. Le modèle Git rend les upgrades et les

rollbacks rapides, mais exige de la rigueur dans la gestion de la configuration et des backups.

Les clés du succès : un environnement de staging, une fenêtre de maintenance planifiée, une communication claire avec l'équipe, et une vérification systématique après chaque upgrade. En suivant ces pratiques, les upgrades de PmaControl deviennent une opération de routine — pas une source de stress.