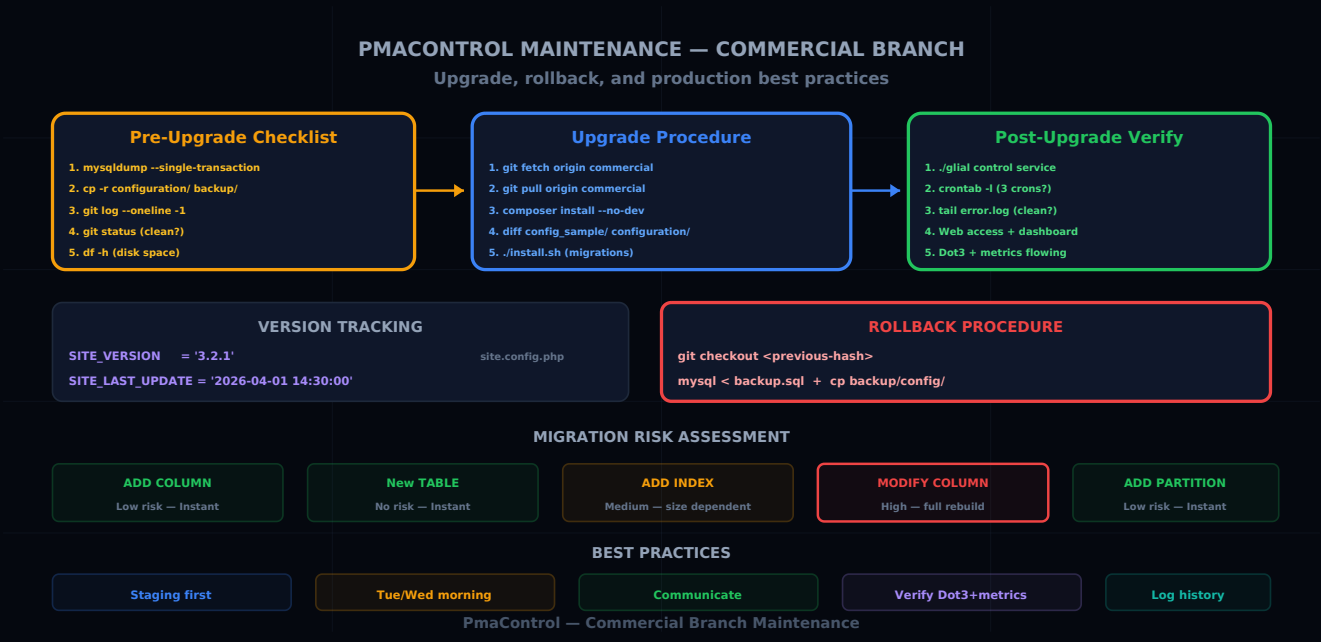
Utrzymanie PmaControl gałąź commercial: aktualizacja, rollback i dobre praktyki

Aurélien LEQUOY · April 13, 2026

PMACONTROL UPGRADE MAINTENANCE GIT PRODUCTION



PmaControl to wdrożenie Git

PmaControl nie jest dystrybuowany jako pakiet .deb czy .rpm. To wdrożenie oparte na Git: klonujesz repozytorium, instalujesz zależności Composer, uruchamiasz skrypt instalacyjny i jest na produkcji.

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

Ten model ma swoje zalety (szybka aktualizacja, łatwy rollback, brak menedżera pakietów) i wady (brak automatycznego zarządzania zależnościami systemowymi, brak standardowych skryptów post-install). Ten przewodnik obejmuje codzienne utrzymanie.

Śledzenie wersji

PmaControl przechowuje swoją wersję w pliku konfiguracyjnym witryny:

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

Te stałe są aktualizowane automatycznie podczas instalacji (`install.sh`). Możesz je sprawdzić:

- Przez interfejs webowy: strona "O programie" lub stopka
- Przez CLI: `grep SITE_VERSION configuration/site.config.php`
- Przez API: `GET /api/v1/status` zwraca wersję

Przed każdą aktualizacją zanotuj bieżącą wersję:

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

Lista kontrolna przed aktualizacją

Przed rozpoczęciem aktualizacji wykonaj tę kompletną listę kontrolną:

1. Backup bazy danych

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

Flaga `--single-transaction` jest kluczowa: gwarantuje spójny backup bez blokowania tabel (InnoDB/RocksDB).

Sprawdź rozmiar dumpa:

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

Pusty lub nienaturalnie mały dump wskazuje na problem.

2. Backup konfiguracji

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

Pliki konfiguracyjne są najważniejsze: `db.config.ini.php` (dane dostępowe), `telegram.php`, `acl.config.ini`, `site.config.php`. Aktualizacja nie powinna ich modyfikować, ale ludzki błąd zdarza się szybko.

3. Zanotowanie bieżącej wersji i commita

```
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');
```

Zachowaj hash commita (tutaj `fe0911d`) — to twój punkt rollbacku.

4. Sprawdzenie stanu Git

```
git status
```

Jeśli są niezacommitowane lokalne zmiany, zdecyduj teraz: zacommitować, schować (stash) czy porzucić. `git pull` z lokalnymi zmianami może generować konflikty.

```
# Jeśli zmiany muszą zostać zachowane
git stash save "pre-upgrade $(date +%Y%m%d)"

# Jeśli muszą zostać porzucone
git checkout -- .
```

5. Sprawdzenie przestrzeni dyskowej

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

Aktualizacja i migracje mogą wymagać przestrzeni tymczasowej. Upewnij się, że masz co najmniej 20% wolnego miejsca na obu partycjach.

Procedura aktualizacji

Krok 1: Pobranie zmian

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

Przed mergem sprawdź, co się zmieniło:

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

To wyświetla wszystkie commity między twoją wersją a najnowszą. Przeczytaj komunikaty commitów, aby zidentyfikować:

- Zmiany schematu bazy danych
- Modyfikacje konfiguracji
- Dodane lub zmodyfikowane zależności Composer
- Zgłoszone breaking changes

Krok 2: Zastosowanie aktualizacji

```
git pull origin commercial
```

Jeśli pojawią się konflikty, prawie zawsze dotyczą plików konfiguracyjnych. Nigdy nie rozwiązuj konfliktu ślepo akceptując "theirs" — sprawdź ręcznie.

Krok 3: Aktualizacja zależności

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

`composer install` (bez `update`) używa `composer.lock` z repozytorium, co gwarantuje te same wersje co zespół deweloperski. **Nigdy** nie używaj `composer update` na produkcji — może pobrać nietestowane wersje.

Sprawdź, czy instalacja przebiegła pomyślnie:

```
# Sprawdź, czy nie ma błędu
echo $?
# Powinno zwrócić 0
```

```
# Sprawdź katalog vendor
ls -la vendor/autoload.php
```

Krok 4: Sprawdzenie zmian konfiguracji

Porównaj pliki przykładowe z bieżącą konfiguracją:

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

Jeśli nowe pliki pojawiły się w `config_sample/`, których nie ma w `configuration/`, to nowe konfiguracje do zintegrowania:

```
# Lista nowych plików w 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
```

Dla każdego nowego pliku:

```
cp config_sample/new_config.php configuration/new_config.php
# Edytuj i dostosuj wartości do swojego środowiska
```

Krok 5: Uruchomienie migracji

```
./install.sh
```

Skrypt `install.sh` zarządza migracjami schematu bazy danych. Wykonuje następujące czynności:

1. Wykrywa aktualną wersję schematu
2. Stosuje brakujące migracje w kolejności
3. Aktualizuje `SITE_VERSION` i `SITE_LAST_UPDATE`

Zalecana weryfikacja ręczna: przed uruchomieniem `install.sh` przejrzyj migracje:

```
# Lista plików migracji
ls -la data/migrations/ 2>/dev/null || ls -la install/migrations/ 2>/dev/null
```

Jeśli migracja zawiera `ALTER TABLE` na dużych tabelach (np. `ts_value_general_int`), zaplanuj dłuższe okno konserwacji.

Lista kontrolna po aktualizacji

1. Restart usług

```
./glial control service
```

To polecenie restartuje cykl zbierania i przetwarzania danych. Sprawdź, czy nie zwraca błędów.

2. Weryfikacja cronów

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

Sprawdź, czy trzy niezbędne crony są obecne:

```
* * * * * 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. Sprawdzenie logów błędów

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

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

Szukaj błędów krytycznych, ostrzeżeń o brakujących klasach, błędów bazy danych. Udana aktualizacja nie powinna generować żadnych nowych błędów.

4. Weryfikacja dostępu webowego

Otwórz PmaControl w przeglądarce i sprawdź:

- Strona logowania działa
- Główny dashboard się ładuje
- Lista serwerów się wyświetla
- Pojedynczy serwer jest dostępny
- Strona slave działa (jeśli dotyczy)
- Topologia Dot3 się ładuje

5. Weryfikacja metryk

Poczekaj 5 minut (pełny cykl zbierania danych), a następnie sprawdź:

```
# Czy agenci zbierają dane?
./glial agent check_daemon

# Czy dane napływają?
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;
"
```

Jeśli to zapytanie zwraca wyniki, niektóre serwery nie otrzymują danych — zbadaj sprawę.

6. Weryfikacja Dot3 i topologii

```
./glial dot3 generate
```

Sprawdź, czy topologia regeneruje się bez błędów i czy graf w interfejsie webowym jest spójny.

Procedura rollback

Jeśli coś pójdzie nie tak, oto jak cofnąć zmiany.

Rollback kodu

```
cd /srv/www/pmacontrol
git checkout <poprzedni-hash-commita>
```

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

Na przykład, jeśli commit przed aktualizacją to `fe0911d`:

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

Rollback bazy danych (jeśli schemat się zmienił)

Jeśli `install.sh` zmodyfikował schemat, trzeba przywrócić backup:

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

Uwaga: ta operacja nadpisuje wszystkie dane zebrane od momentu backupu. Jeśli aktualizacja miała miejsce 2 godziny temu, tracisz 2 godziny metryk. Dlatego aktualizację należy planować w oknie konserwacji.

Rollback konfiguracji

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

Po rollbacku

```
./glial control service
# Sprawdź logi
tail -20 /tmp/pmacontrol_agent.log
# Sprawdź dostęp webowy
curl -s -o /dev/null -w "%{http_code}" https://pmacontrol.example.com/
# Powinno zwrócić 200
```

Zarządzanie zależnościami Composer

Zrozumienie lock file

Plik `composer.lock` jest wersjonowany w repozytorium. Gwarantuje, że wszyscy używają dokładnie tych samych wersji zależności. Gdy `composer.lock` zmienia się w pullu:

```
# Zobacz zmiany zależności
git diff HEAD~1 composer.lock | grep '"name"'
```

Sprawdzanie breaking changes

Jeśli ważna zależność się zmienia (np. framework Glial z 2.x na 3.x), to ryzykowna zmiana. Sprawdź CHANGELOG tej zależności:

```
# Lista zaktualizowanych zależności
composer show --latest --outdated
```

Nigdy composer update na produkcji

```
# NIE — pobiera najnowsze możliwe wersje
composer update

# TAK — instaluje dokładnie wersje z lock file
composer install --no-dev
```

Migracje bazy danych

Jak działają

`install.sh` wykonuje migracje sekwencyjnie. Każda migracja jest idempotentna — najpierw sprawdza, czy modyfikacja została już zastosowana:

```
-- Przykład typowej migracji
-- Sprawdza, czy kolumna istnieje przed jej dodaniem
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;
```

Ryzykowne migracje

Niektóre migracje są bardziej ryzykowne niż inne:

Typ	Ryzyko	Czas
ADD COLUMN (nullable)	Niskie	Natychmiastowy (MariaDB 10.0+)
ADD INDEX	Średnie	Proporcjonalny do rozmiaru
MODIFY COLUMN (zmiana typu)	Wysokie	Pełna przebudowa tabeli
DROP COLUMN	Niskie	Natychmiastowy (MariaDB 10.4+)
ADD PARTITION	Niskie	Natychmiastowy
Nowa tabela	Brak	Natychmiastowy

Dla dużych tabel (takich jak `ts_value_general_int`, która może mieć kilka GB), `ADD INDEX` może trwać minuty, a nawet godziny. Planuj odpowiednio.

Zalecany przegląd ręczny

Przed uruchomieniem `install.sh` przeczytaj pliki migracji, aby zrozumieć, co zostanie zmodyfikowane. Jeśli migracja wydaje ci się ryzykowna (ALTER TABLE na tabeli kilku-gigabajtowej), przetestuj ją najpierw na środowisku staging.

Dobre praktyki

1. Środowisko staging

Utrzymuj środowisko staging, które replikuje twoją produkcję. Aktualizacja jest tam testowana przed zastosowaniem na produkcji:

```
1. Staging: git pull → composer install → install.sh → weryfikacja
2. Odczekaj 24h — obserwuj logi
3. Produkcja: ta sama procedura
```

Staging nie musi nadzorować tylu serwerów co produkcja. 5-10 serwerów MariaDB / MySQL wystarczy do zwalidowania poprawnego działania.

2. Planowane okno konserwacji

Nigdy nie aktualizuj w piątek o 17:00. Zaplanuj:

- **Wtorek lub środa**: w środku tygodnia, zespół jest dostępny
- **Rano**: aby mieć cały dzień na wykrycie problemów
- **Poza szczytem**: nie podczas wdrożenia aplikacji ani nocnego batcha

3. Komunikuj

Powiadom zespół przed aktualizacją:

```
Temat: Konserwacja PmaControl — Wt 13 kwietnia 09:00-10:00

Instancja PmaControl zostanie zaktualizowana z wersji 3.2.1 do 3.3.0.
Podczas konserwacji (ok. 30 minut):
- Dashboardy mogą być tymczasowo niedostępne
- Zbieranie metryk zostanie przerwane (automatyczne nadrobienie)
- Alerty Telegram zostaną wstrzymane, a następnie wznowione

Kontakt: dba@company.com
```

4. Zweryfikuj Dot3 + metryki po aktualizacji

To najważniejszy test dymny. Jeśli metryki napływają i topologia jest poprawna, aktualizacja się powiodła. Jeśli jedno z nich nie działa, jest problem do zbadania.

5. Prowadź historię aktualizacji

Utrzymuj prosty plik z listą wykonanych aktualizacji:

```
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 — wolna migracja ts_value (45min)
2026-02-01 08:30 | 3.0.5 → 3.1.0 | 9876def → 1234abc | ROLLBACK — bug #342 w Listener
```

6. Automatyzuj, gdy jest gotowe

Gdy wykonasz 5-10 ręcznych aktualizacji bez incydentów, możesz zautomatyzować skryptem:

```
#!/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"

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

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

echo "Upgrade successful"
```

Ale zawsze zachowaj możliwość ręcznego rollbacku.

Najczęstsze błędy

"Class not found" po aktualizacji

Objaw: błąd PHP `Class 'Xyz' not found` w logach.

Przyczyna: `composer install` nie został wykonany po pullu, lub autoloader nie został zregenerowany.

Rozwiązanie:

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

Błąd migracji "Table already exists"

Objaw: `install.sh` kończy się błędem `Table 'xyz' already exists`.

Przyczyna: migracja nie jest idempotentna (błąd w skrypcie migracji).

Rozwiązanie: sprawdź ręcznie, czy tabela/kolumna istnieje i pomiń migrację, jeśli to konieczne. Zgłoś błąd zespołowi PmaControl.

Konflikty Git w konfiguracji

Objaw: `git pull` kończy się błędem z konfliktami w `configuration/`.

Przyczyna: zmodyfikowałeś plik konfiguracyjny, który został również zmodyfikowany upstream.

Rozwiązanie:

```
# Zapisz swoją wersję
cp configuration/problematic_file.php /tmp/

# Zaakceptuj wersję upstream
git checkout --theirs configuration/problematic_file.php
git add configuration/problematic_file.php

# Ponownie zastosuj swoje modyfikacje ręcznie
# Porównaj /tmp/problematic_file.php z wersją upstream
```

Stash jest zagubiony

Objaw: wykonałeś `git stash` przed aktualizacją i nie możesz znaleźć swoich zmian.

Rozwiązanie:

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

git stash pop stash@{0}
```

Podsumowanie

Utrzymanie PmaControl na produkcji to przewidywalny proces, jeśli postępujesz zgodnie z procedurą: backup, pull, composer install, install.sh, weryfikacja. Model Git sprawia, że aktualizacje i rollbacki są szybkie, ale wymaga rygorystycznego zarządzania konfiguracją i backupami.

Klucze do sukcesu: środowisko staging, planowane okno konserwacji, jasna komunikacja z zespołem i systematyczna weryfikacja po każdej aktualizacji. Stosując te praktyki, aktualizacje

PmaControl stają się rutynową operacją — a nie źródłem stresu.