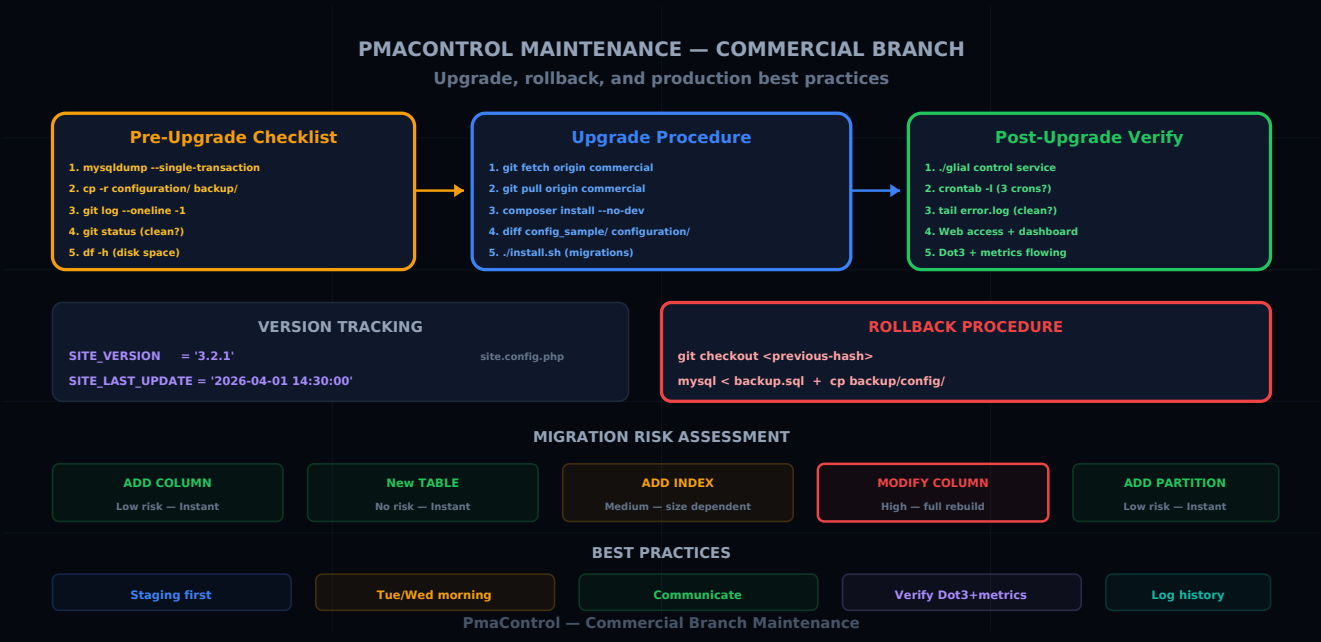
Обслуживание ветки commercial PmaControl: обновление, откат и лучшие практики

Aurélien LEQUOY · April 13, 2026

PMACONTROL UPGRADE MAINTENANCE GIT PRODUCTION



PmaControl — это Git-деплой

PmaControl не распространяется в виде пакета .deb или .rpm. Это деплой на основе Git: вы клонируете репозиторий, устанавливаете зависимости Composer, запускаете скрипт установки — и это в продакшене.

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

У этой модели есть преимущества (быстрое обновление, лёгкий откат, без менеджера пакетов) и недостатки (нет автоматического управления системными зависимостями, нет стандартизированных post-install скриптов). Данное руководство охватывает повседневное обслуживание.

Отслеживание версий

PmaControl хранит свою версию в конфигурационном файле сайта:

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

Эти константы обновляются автоматически при установке (`install.sh`). Проверить их можно:

- Через веб-интерфейс: страница «О программе» или подвал
- Через CLI: `grep SITE_VERSION configuration/site.config.php`
- Через API: `GET /api/v1/status` возвращает версию

Перед любым обновлением запишите текущую версию:

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

Чек-лист перед обновлением

Перед запуском обновления выполните полный чек-лист:

1. Резервное копирование базы данных

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

Флаг `--single-transaction` критически важен: он гарантирует целостную резервную копию без блокировки таблиц (InnoDB/RocksDB).

Проверьте размер дампа:

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

Пустой или аномально маленький дамп указывает на проблему.

2. Резервное копирование конфигурации

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

Файлы конфигурации наиболее критичны: `db.config.ini.php` (учётные данные), `telegram.php`, `acl.config.ini`, `site.config.php`. Обновление не должно их изменять, но человеческая ошибка случается быстро.

3. Запишите текущую версию и коммит

```
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');
```

Сохраните хеш коммита (здесь `fe0911d`) — это ваша точка отката.

4. Проверьте состояние Git

```
git status
```

Если есть незакоммиченные локальные изменения, решите сейчас: закоммитить, застэшить или отбросить. `git pull` с локальными изменениями может вызвать конфликты.

```
# Если изменения нужно сохранить
git stash save "pre-upgrade $(date +%Y%m%d)"

# Если их нужно отбросить
git checkout -- .
```

5. Проверьте дисковое пространство

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

Обновление и миграции могут потребовать временное пространство. Убедитесь, что свободно не менее 20% на обоих разделах.

Процедура обновления

Шаг 1: Получите изменения

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

Перед слиянием изучите, что изменилось:

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

Это покажет все коммиты между вашей версией и последней. Прочитайте сообщения коммитов, чтобы определить:

- Изменения схемы базы данных
- Модификации конфигурации
- Добавленные или изменённые зависимости Composer
- Отмеченные breaking changes

Шаг 2: Примените обновление

```
git pull origin commercial
```

Если появятся конфликты, они почти всегда будут в файлах конфигурации. Никогда не разрешайте конфликт слепо, принимая «theirs» — проверяйте вручную.

Шаг 3: Обновите зависимости

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

`composer install` (без `update`) использует `composer.lock` из репозитория, что гарантирует те же версии, что и у команды разработки. **Никогда** не используйте `composer update` в продакшене — это может подтянуть непротестированные версии.

Убедитесь, что установка прошла успешно:

```
# Проверьте отсутствие ошибок
echo $?
# Должно вернуть 0

# Проверьте папку vendor
ls -la vendor/autoload.php
```

Шаг 4: Проверьте изменения конфигурации

Сравните файлы-образцы с вашей текущей конфигурацией:

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

Если в `config_sample/` появились новые файлы, которых нет в `configuration/`, это новые конфигурации для интеграции:

```
# Перечислить новые файлы в 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
```

Для каждого нового файла:

```
cp config_sample/new_config.php configuration/new_config.php
# Отредактировать и адаптировать значения под вашу среду
```

Шаг 5: Выполните миграции

```
./install.sh
```

Скрипт `install.sh` управляет миграциями схемы базы данных. Он:

1. Определяет текущую версию схемы
2. Применяет недостающие миграции по порядку
3. Обновляет `SITE_VERSION` и `SITE_LAST_UPDATE`

Рекомендуемая ручная проверка: перед запуском `install.sh` изучите миграции:

```
# Вывести файлы миграций
ls -la data/migrations/ 2>/dev/null || ls -la install/migrations/ 2>/dev/null
```

Если миграция содержит `ALTER TABLE` на объёмных таблицах (как `ts_value_general_int`), предусмотрите более длительное окно обслуживания.

Чек-лист после обновления

1. Перезапустите сервисы

```
./glial control service
```

Эта команда перезапускает цикл сбора и обработки данных. Убедитесь, что она не возвращает ошибок.

2. Проверьте cron-задачи

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

Убедитесь, что три основных cron-задачи присутствуют:

```
* * * * * 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. Проверьте логи ошибок

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

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

Ищите фатальные ошибки, предупреждения об отсутствующих классах, ошибки базы данных. Успешное обновление не должно генерировать новых ошибок.

4. Проверьте веб-доступ

Откройте PmaControl в браузере и проверьте:

- Страница входа работает
- Главная панель загружается
- Список серверов отображается
- Отдельный сервер доступен
- Страница slave работает (если применимо)
- Топология Dot3 загружается

5. Проверьте метрики

Подождите 5 минут (полный цикл сбора), затем проверьте:

```
# Собирают ли агенты данные?
./glial agent check_daemon

# Приходят ли данные?
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;
"
```

Если этот запрос возвращает результаты, некоторые серверы больше не получают данных — необходимо расследование.

6. Проверьте Dot3 и топологию

```
./glial dot3 generate
```

Убедитесь, что топология генерируется без ошибок и граф в веб-интерфейсе корректен.

Процедура отката

Если что-то пошло не так, вот как вернуться назад.

Откат кода

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

Например, если коммит до обновления был `fe0911d`:

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

Откат базы данных (если схема изменилась)

Если `install.sh` изменил схему, нужно восстановить резервную копию:

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

Внимание: эта операция перезаписывает все данные, собранные с момента бэкапа. Если обновление было 2 часа назад, вы потеряете 2 часа метрик. Поэтому обновление должно планироваться во время окна обслуживания.

Откат конфигурации

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

После отката

```
./glial control service
# Проверьте логи
tail -20 /tmp/pmacontrol_agent.log
# Проверьте веб-доступ
curl -s -o /dev/null -w "%{http_code}" https://pmacontrol.example.com/
# Должно вернуть 200
```

Управление зависимостями Composer

Понимание lock-файла

Файл `composer.lock` версионируется в репозитории. Он гарантирует, что все используют точно такие же версии зависимостей. Когда `composer.lock` меняется при pull:

```
# Посмотреть изменения зависимостей
git diff HEAD~1 composer.lock | grep '"name"'
```

Проверка breaking changes

Если мажорная зависимость изменяется (например, Glial framework с 2.x на 3.x), это рискованное изменение. Проверьте CHANGELOG зависимости:

```
# Вывести обновлённые зависимости
composer show --latest --outdated
```

Никогда не делайте composer update в продакшене

```
# НЕТ — подтягивает последние возможные версии
composer update

# ДА — устанавливает точно версии из lock-файла
composer install --no-dev
```

Миграции базы данных

Как они работают

`install.sh` выполняет миграции последовательно. Каждая миграция идемпотентна — она сначала проверяет, было ли изменение уже применено:

```
-- Пример типичной миграции
-- Проверяет существование колонки перед добавлением
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;
```

Рискованные миграции

Некоторые миграции более рискованны, чем другие:

Тип	Риск	Время
ADD COLUMN (nullable)	Низкий	Мгновенно (MariaDB 10.0+)
ADD INDEX	Средний	Пропорционально размеру
MODIFY COLUMN (смена типа)	Высокий	Полная перестройка таблицы
DROP COLUMN	Низкий	Мгновенно (MariaDB 10.4+)
ADD PARTITION	Низкий	Мгновенно
Новая таблица	Нет	Мгновенно

Для объёмных таблиц (таких как `ts_value_general_int`, которая может занимать несколько ГБ), `ADD INDEX` может занять минуты или даже часы. Планируйте соответственно.

Рекомендуемая ручная проверка

Перед запуском `install.sh` прочитайте файлы миграций, чтобы понять, что будет изменено. Если миграция кажется рискованной (ALTER TABLE на таблице из нескольких ГБ), сначала протестируйте её в среде staging.

Лучшие практики

1. Среда staging

Поддерживайте среду staging, которая воспроизводит ваш продакшен. Обновление тестируется там перед применением в продакшене:

```
1. Staging: git pull → composer install → install.sh → проверка
2. Подождать 24 часа — наблюдать за логами
3. Production: та же процедура
```

Staging не нуждается в мониторинге того же количества серверов, что и продакшен. 5-10 серверов MariaDB / MySQL достаточно для проверки работоспособности.

2. Запланированное окно обслуживания

Никогда не делайте обновление в пятницу в 17:00. Планируйте:

- **Вторник или среда**: середина недели, команда на месте
- **Утро**: чтобы иметь целый день для обнаружения проблем
- **Вне пика**: не во время деплоя приложений или ночных батчей

3. Коммуникация

Предупредите команду перед обновлением:

```
Тема: Обслуживание PmaControl — Вт 13 апреля 09:00-10:00

Экземпляр PmaControl будет обновлён с версии 3.2.1 до 3.3.0.
Во время обслуживания (примерно 30 минут):
- Дашборды могут быть временно недоступны
- Сбор метрик будет прерван (автоматическое восстановление)
- Алерты Telegram будут приостановлены, затем возобновлены

Контакт: dba@company.com
```

4. Проверяйте Dot3 + метрики после обновления

Это самый важный smoke-тест. Если метрики поступают и топология корректна, обновление успешно. Если одно из двух не работает, есть проблема для расследования.

5. Ведите историю обновлений

Поддерживайте простой файл с перечнем выполненных обновлений:

```
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 — медленная миграция ts_value
(45мин)
2026-02-01 08:30 | 3.0.5 → 3.1.0 | 9876def → 1234abc | ОТКАТ — баг #342 в Listener
```

6. Автоматизируйте, когда процесс созреет

Когда вы выполните 5-10 ручных обновлений без инцидентов, можно автоматизировать скриптом:

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

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

# Бэкап
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"

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

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

echo "Upgrade successful"
```

Но всегда сохраняйте возможность ручного отката.

Типичные ошибки

"Class not found" после обновления

Симптом: ошибка PHP `Class 'Xyz' not found` в логах.

Причина: `composer install` не был выполнен после pull, или автозагрузчик не был перегенерирован.

Решение:

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

Ошибка миграции "Table already exists"

Симптом: `install.sh` завершается с ошибкой `Table 'xyz' already exists`.

Причина: миграция не идемпотентна (баг в скрипте миграции).

Решение: проверьте вручную, существует ли таблица/колонка, и пропустите миграцию при необходимости. Сообщите о баге команде PmaControl.

Git-конфликты в конфигурации

Симптом: `git pull` завершается с конфликтами в `configuration/`.

Причина: вы изменили файл конфигурации, который также был изменён upstream.

Решение:

```
# Сохраните вашу версию
cp configuration/problematic_file.php /tmp/

# Примите upstream-версию
git checkout --theirs configuration/problematic_file.php
git add configuration/problematic_file.php

# Повторно примените ваши изменения вручную
# Сравните /tmp/problematic_file.php с upstream-версией
```

Потерянный stash

Симптом: вы сделали `git stash` перед обновлением и не можете найти свои изменения.

Решение:

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

git stash pop stash@{0}
```

Заключение

Обслуживание PmaControl в продакшене — предсказуемый процесс, если вы следуете процедуре: бэкап, pull, composer install, install.sh, проверка. Модель Git делает обновления и откаты быстрыми, но требует тщательности в управлении конфигурацией и бэкапами.

Ключи к успеху: среда staging, запланированное окно обслуживания, ясная коммуникация с командой и систематическая проверка после каждого обновления. Следуя этим практикам, обновления PmaControl становятся рутинной операцией — а не источником стресса.