GIT GITHUB версии и синхронизация

Выравнивание production -> github

Инструкция: первичная синхронизация ядра CRM Voronka с GitHub

Цель

Создать чистую базовую точку разработки для ядра CRM Voronka:

```
production core = dev core = GitHub baseline
```

Рабочие каталоги:

```
Production:
/opt/voronkacrm/core

Development:
/opt/voronkacrmdev/core

GitHub:
git@github.com:voronkaapp/core_crm_voronka_pro.git
```

В production контейнеры клиентов монтируют ядро как:

```
/voronka
```

Dev-контейнер должен монтировать:

```
/opt/voronkacrmdev/core → /voronka
```

---

1. Что важно перед началом

Нельзя выполнять:

```
git reset --hard
git clean -fd
git pull
```

до создания backup, если не подтверждено, что текущая папка полностью безопасна для очистки.

Нельзя хранить старую `.git`-папку внутри проекта под именем `.git.old_*`, потому что она может случайно попасть в новый commit.

Правильно:

```
старую .git переносить за пределы проекта
```

Например:

```
/opt/git-backups/
```

---

2. Сделать backup старого GitHub-репозитория

```
mkdir -p /opt/git-backups
cd /opt/git-backups
```

```
git clone --mirror git@github.com:voronkaapp/core_crm_voronka_pro.git \
  core_crm_voronka_pro_old_$(date +%Y%m%d_%H%M%S).git
```

Если доступ к GitHub по SSH ещё не настроен, сначала настроить SSH deploy key или SSH key пользователя.

---

3. Сделать backup production и dev папок

```
mkdir -p /opt/core-backups

tar -czf /opt/core-backups/prod-core-before-git-$(date +%Y%m%d_%H%M%S).tar.gz \
  -C /opt/voronkacrm core

tar -czf /opt/core-backups/dev-core-before-git-$(date +%Y%m%d_%H%M%S).tar.gz \
  -C /opt/voronkacrmdev core
```

---

4. Подготовить production-каталог как источник baseline

Перейти в production-каталог:

```
cd /opt/voronkacrm/core
```

Проверить наличие `.git`:

```
ls -la .git
```

Если `.git` есть, перенести её за пределы проекта:

```
mkdir -p /opt/git-backups

mv .git /opt/git-backups/core_git_old_$(date +%Y%m%d_%H%M%S)
```

Важно: не делать так:

```
mv .git .git.old_20260705_062203
```

Такую папку можно случайно добавить в новый Git. В ней могут быть большие pack-файлы больше 100 MB, из-за чего GitHub отклонит push.

---

5. Создать или проверить `.gitignore`

Файл:

```
nano /opt/voronkacrm/core/.gitignore
```

Минимальное содержимое:

```
# IDE
/.idea/
/.vscode/

# OS
.DS_Store
Thumbs.db

# Environment / secrets
/.env
.env.*
!/env.example

# Runtime/cache/logs
/cache/*
```

```
!/cache/.gitkeep
/logs/*
/tmp/*
/runtime/*
/storage/*

# CRM generated/user data
/user_privileges/*
/upload/*
/uploads/*

# Old local Git backups - must never be committed
.git.old_*/
.git.old*/
/.git-backup*/
/git-backup*/

# Logs/backups/temp
*.log
*.bak
*.backup
*.old
*.orig
*.swp
```

Если проект содержит рабочий `vendor/`, который не устанавливается автоматически через Composer на сервере, не добавлять `/vendor/` в `.gitignore`.

---

6. Инициализировать новый Git в production-каталоге

```
cd /opt/voronkacrm/core


git init -b main
git remote add origin git@github.com:voronkaapp/core_crm_voronka_pro.git
```

Если remote уже существует:

```
git remote set-url origin git@github.com:voronkaapp/core_crm_voronka_pro.git
```

Проверить:

```
git remote -v
```

Ожидаемо:

```
origin  git@github.com:voronkaapp/core_crm_voronka_pro.git (fetch)
origin  git@github.com:voronkaapp/core_crm_voronka_pro.git (push)
```

---

7. Проверить, что попадёт в Git

Сначала сухая проверка:

```
git add -n .
```

Потом добавить файлы в индекс:

```
git add -A
```

Проверить опасные файлы:

```
git status --short |  grep -Ei
'env|password|secret|token|dump|backup|sql|log|upload|user_privileges|cache|storage|runtime|git.old'
```

Если выводит подозрительные файлы, остановиться и поправить `.gitignore`.

Если лишние файлы уже попали в индекс, удалить их из индекса:

```
git rm -r --cached cache logs upload uploads user_privileges storage runtime 2>/dev/null ||
true
git rm -r --cached .git.old_* 2>/dev/null || true
```

Проверить, что старые `.git.old` не отслеживаются:

```
git ls-files | grep '^\.git.old' || echo "OK: .git.old is not tracked"
```

Проверить крупные файлы больше 90 MB:

```
find . -type f -size +90M \
  -not -path './.git/*' \
  -not -path './cache/*' \
  -not -path './logs/*' \
  -not -path './upload/*' \
  -not -path './uploads/*' \
  -not -path './storage/*' \
  -not -path './runtime/*' \
  -print
```

Если команда что-то вывела, проверить каждый файл. GitHub не принимает обычные файлы больше 100 MB.

---

8. Создать baseline commit

```
cd /opt/voronkacrm/core


git commit -m "Baseline current production Voronka CRM core"
```

Создать метку:

```
git tag -a baseline-20260705-core-current \
  -m "Baseline: current production and dev core before new workflow"
```

Проверить:

```
git log --oneline -3
git tag --points-at HEAD
git status --short
```

Ожидаемо:

```
baseline-20260705-core-current
```

А:

```
git status --short
```

должен быть пустым.

---

9. Отправить baseline в GitHub

```
git push --force origin main
git push --force origin baseline-20260705-core-current
```

Если GitHub отклоняет push из-за большого файла, пример ошибки:

```
File .git.old_.../objects/pack/pack-....pack is 284.12 MB
GH001: Large files detected
```

Значит старая `.git.old_*` была добавлена в commit. Исправление:

```
cd /opt/voronkacrm/core

mkdir -p /opt/git-backups
mv .git.old_* /opt/git-backups/ 2>/dev/null || true

cat >> .gitignore <<'EOF'

# Old local Git backups
.git.old_*/
```

```
.git.old*/
EOF

git rm -r --cached .git.old_* 2>/dev/null || true
git add .gitignore
git commit --amend -m "Baseline current production Voronka CRM core"

git tag -d baseline-20260705-core-current
git tag -a baseline-20260705-core-current \
  -m "Baseline: current production and dev core before new workflow"

git push --force origin main
git push --force origin baseline-20260705-core-current
```

---

10. Привести dev-папку к той же метке

Перейти в dev-каталог:

```
cd /opt/voronkacrmdev/core
```

Если там есть `.git`, перенести за пределы проекта:

```
mkdir -p /opt/git-backups

mv .git /opt/git-backups/dev_core_git_old_$(date +%Y%m%d_%H%M%S) 2>/dev/null || true
```

Инициализировать Git:

```
git init -b main
git remote add origin git@github.com:voronkaapp/core_crm_voronka_pro.git
git fetch origin --tags
git reset --hard baseline-20260705-core-current
```

Проверить untracked-файлы:

```
git clean -fdn
```

Если выводит только мусор, который не нужен, можно удалить:

```
git clean -fd
```

Если выводит конфиги, upload, cache, пользовательские данные — сначала проверить `.gitignore`.

---

11. Проверить, что production и dev одинаковые

Production:

```
cd /opt/voronkacrm/core

git rev-parse HEAD
git tag --points-at HEAD
git status --short
```

Dev:

```
cd /opt/voronkacrmdev/core

git rev-parse HEAD
git tag --points-at HEAD
git status --short
```

Ожидаемо:

```
одинаковый commit hash
одинаковая метка baseline-20260705-core-current
git status --short пустой
```

12. Рабочий процесс после baseline

Разработка ведётся только в dev-каталоге:

```
/opt/voronkacrmdev/core
```

Production-каталог руками не редактировать:

```
/opt/voronkacrm/core
```

Перед началом задачи:

```
cd /opt/voronkacrmdev/core

git checkout main
git pull --ff-only origin main
git checkout -b feature/task-name
```

После доработки:

```
git status
git diff
git add -A
git commit -m "Описание изменения"
git push origin feature/task-name
```

После проверки можно слить в `main` и создать release tag:

```
git checkout main
git pull --ff-only origin main
git merge feature/task-name

git tag -a release-YYYYMMDD-001 -m "Release YYYYMMDD-001"

git push origin main
```

```
git push origin release-YYYYMMDD-001
```

---

13. Обновление production после релиза

Перед обновлением production:

```
cd /opt/voronkacrm/core


git status --short
```

Если вывод не пустой, deployment остановить и разобраться.

Если чисто:

```
git fetch origin --tags
git checkout release-YYYYMMDD-001
```

После обновления сбросить opcache:

```
docker exec crm-prod-php php -r 'opcache_reset();'
```

Если имя контейнера отличается, подставить реальное имя PHP-контейнера.

---

14. Правило для миграций баз данных

Если релиз требует изменений структуры или данных БД:

```
1. Сделать backup всех клиентских баз.
2. Запустить миграции на тестовой базе.
3. Запустить миграции на production-базах клиентов.
```

```
4. Проверить лог выполнения по каждой базе.
5. Только после успешных миграций обновлять production-код.
```

Нельзя выкладывать код, который ожидает новую таблицу или колонку, если миграции ещё не прошли на всех клиентских базах.

---

15. Аварийный hotfix production

Production желательно не редактировать. Если аварийная правка всё же сделана в production:

```
cd /opt/voronkacrm/core

git checkout -b hotfix/prod-issue-YYYYMMDD
git add -A
git commit -m "Hotfix production issue"
git push origin hotfix/prod-issue-YYYYMMDD
```

Потом обязательно подтянуть hotfix в dev:

```
cd /opt/voronkacrmdev/core

git fetch origin
git checkout main
git merge origin/hotfix/prod-issue-YYYYMMDD
```

---

16. Запрещено

Запрещено:

```
редактировать production через IDE
включать auto-upload на production
```

```
хранить .git.old_* внутри проекта
делать git reset --hard без backup
пушить cache/logs/upload/user_privileges/storage/runtime
пушить .env, пароли, токены, дампы баз
выкладывать код до успешных миграций БД
```

---

17. Контрольные команды

Проверить текущий commit:

```
git rev-parse HEAD
```

Проверить метку на текущем commit:

```
git tag --points-at HEAD
```

Проверить чистоту рабочей папки:

```
git status --short
```

Проверить remote:

```
git remote -v
```

Проверить крупные файлы:

```
find . -type f -size +90M \
  -not -path './.git/*' \
  -not -path './cache/*' \
  -not -path './logs/*' \
  -not -path './upload/*' \
  -not -path './uploads/*' \
  -not -path './storage/*' \
  -not -path './runtime/*' \
  -print
```

Проверить, что `.git.old` не попала в Git:

```
git ls-files | grep '^\.git.old' || echo "OK: .git.old is not tracked"
```

Итог

После выполнения этой инструкции должны быть выполнены условия:

1. GitHub содержит актуальное ядро CRM.
2. Production и dev находятся на одном commit.
3. На текущем commit есть baseline tag.
4. Runtime-файлы, кеши, upload, user_privileges, секреты и старая .git история не попали в Git.
5. Дальнейшая разработка ведётся только в dev-папке.
6. Production обновляется только через GitHub release tag.