

Документация Adaptadocx

0.1.0

Содержание

Adaptadocx — это система публикации документации на базе Antora. Платформа автоматически собирает многоязычные материалы из исходников AsciiDoc и создаёт три вида артефактов: **HTML**, **PDF** и **DOCX**. В проект встроены проверки качества, аудит безопасности и CI-конвейеры для воспроизводимых сборок.

Система поддерживает:

* две локали (английская и русская);
* кастомизацию UI Antora;
* рендеринг диаграмм **SVG**;
* сборки в Docker;
* автоматизированные процессы **QA**, **Security** и линтинг;
* мультиверсии по Git-тегам.

# Обзор архитектуры

Adaptadocx построен модульно: Antora служит ядром генерации, а вспомогательные компоненты расширяют её возможности.

| Компонент | Назначение |
| --- | --- |
| Исходники документации | Контент AsciiDoc для EN и RU в docs/ |
| Конфигурация | Тема PDF и метаданные локалей в config/ |
| UI-компоненты | Кастомизация Antora UI в custom-ui/ |
| Скрипты сборки | Makefile и альтернативный оркестратор build.py |
| CI/CD | Workflow GitHub Actions в .github/ |

# Ключевые возможности

## Мультиформатный вывод

Единый набор AsciiDoc-файлов порождает три независимых артефакта.

* **HTML** — статический сайт с кастомным UI-bundle и полнотекстовым поиском
* **PDF** — офлайн/печать с пользовательской темой и шрифтами DejaVu для кириллицы
* **DOCX** — редактируемый документ через Pandoc с автоматической обложкой

## Контроль качества и безопасность

Проверки запускаются автоматически в CI.

* **Vale** — линтер AsciiDoc (vale.xml)
* **htmltest** — проверка ссылок (htmltest.log)
* **Shellcheck** — анализ Bash-скриптов
* **Security audit** — OSV-Scanner, Sandworm и поиск запрещённых паттернов (неблокирующие)

# Конвейер сборки

Adaptadocx можно собирать **Make** (по умолчанию) или **эквивалентным Python-скриптом**. Оба варианта формируют одинаковые артефакты в build/.

## Режимы сборки

**Локальный** (по умолчанию) — собирает только текущую ветку HEAD или ветку, указанную в BUILD\_REF.

make build-all  
make build-all BUILD\_REF=my-feature

**По тегам** — мультиверсийная сборка по всем Git-тегам.

make build-all BUILD\_SCOPE=tags

Выходные артефакты версионированы и размещаются в:

* build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* build/docx/<locale>/<version>/adaptadocx-<locale>.docx
* копии публикуются в site/<locale>/<version>/\_downloads/

Ссылки **PDF** и **DOCX** в шапке всегда указывают на \_downloads активной версии.

## Docker-окружение

Контейнерный образ включает:

* Node.js 20 и Ruby ≥ 2.7
* Python 3.11+
* Asciidoctor PDF
* Vale, htmltest, Shellcheck
* Graphviz
* Шрифты DejaVu
* rsvg-convert для конвертации **SVG**→**PDF**/**PNG**

## Цели Makefile

| Цель | Назначение |
| --- | --- |
| make build-all | Собрать **HTML + PDF + DOCX** (алиас build-site) |
| make build-html | Собрать только HTML |
| make build-pdf | Собрать только PDF |
| make build-docx | Собрать только DOCX |
| make test | Запустить Vale, htmltest и Shellcheck |
| make clean | Удалить build/ |
| make release | Заархивировать артефакты после QA |

## Python-скрипт сборки

Если make недоступен, используйте build.py (Python 3.11+).

| Команда | Назначение |
| --- | --- |
| python3 build.py build-site | HTML + PDF + DOCX |
| python3 build.py build-html | Только HTML |
| python3 build.py build-pdf | Только PDF |
| python3 build.py build-docx | Только DOCX |
| python3 build.py prep | Только подстановка версии |
| python3 build.py clean | Удалить build/ |

Используйте либо Makefile, либо Python-скрипт — совмещать их в одной сборке не требуется.

## Непрерывная интеграция

В GitHub Actions определены три группы workflow.

1. **QA Checks** — линтинг AsciiDoc, проверка ссылок, анализ скриптов
2. **Security Audit** — аудит зависимостей и содержимого (OSV-Scanner, Sandworm, запрещённые паттерны)
3. **Release** — полная мультиверсийная сборка (BUILD\_SCOPE=tags), упаковка и деплой

# Быстрый старт

## Установка через Docker (рекомендуется)

docker build -t adaptadocx .  
# сборка Makefile  
docker run --rm -v "$(pwd)":/work adaptadocx make build-all  
# сборка Python-скриптом  
docker run --rm -v "$(pwd)":/work adaptadocx python3 build.py build-site

## Локальная установка

npm ci --no-audit --no-fund  
# Makefile  
make build-all  
# или Python  
python3 build.py build-site

# См. также

* [Быстрый старт](#quick-start)
* [Установка и настройка](#installation)

# Быстрый старт

Следуйте этим шагам, чтобы собрать Adaptadocx во всех трёх форматах **HTML**, **PDF** и **DOCX** и выполнить проверки качества.

## Предварительные требования

Выполните [Установку и настройку](#installation).

## Однократная сборка (все форматы)

**Докер (рекомендуется)**

# Только текущая ветка (по умолчанию)  
docker run --rm -v "$(pwd)":/work adaptadocx make build-all  
  
# По всем тегам (мультиверсийная сборка)  
docker run --rm -v "$(pwd)":/work adaptadocx make build-all BUILD\_SCOPE=tags

**Локальная среда**

# Только текущая ветка (по умолчанию)  
make build-all  
  
# По всем тегам (мультиверсийная сборка)  
make build-all BUILD\_SCOPE=tags

Будут созданы версионированные артефакты:

* build/site/<locale>/<version>/ — HTML
* build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf — PDF
* build/docx/<locale>/<version>/adaptadocx-<locale>.docx — DOCX
* site/<locale>/<version>/\_downloads/ — опубликованные загрузки версии

## Проверка артефактов

tree -L 3 build/

Ожидаемый вывод (пример):

build/  
├── site/  
│ ├── en/  
│ │ └── 0.1.2/  
│ └── ru/  
│ └── 0.1.2/  
├── pdf/  
│ ├── en/  
│ │ └── 0.1.2/  
│ └── ru/  
│ └── 0.1.2/  
└── docx/  
 ├── en/  
 │ └── 0.1.2/  
 └── ru/  
 └── 0.1.2/

## Сборка отдельных форматов

**Только HTML**

make build-html

**Только PDF**

make build-pdf  
# Результат: build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf  
# Копия: site/<locale>/<version>/\_downloads/adaptadocx-<locale>.pdf

**Только DOCX**

make build-docx  
# Результат: build/docx/<locale>/<version>/adaptadocx-<locale>.docx  
# Копия: site/<locale>/<version>/\_downloads/adaptadocx-<locale>.docx

## Запуск проверок качества

make test

Запускаются инструменты:

* **Vale** — vale.xml
* **htmltest** — htmltest.log
* **Shellcheck** — вывод в консоль

Просмотр результатов:

cat vale.xml  
cat htmltest.log

## Упаковка релиза

make release

Будет создан архив adaptadocx-docs-<version>.zip в каталоге build/.

## Цикл «правка — сборка — просмотр»

1. Редактируйте .adoc-файлы в docs/en/…​ или docs/ru/…​.
2. Очистите предыдущую сборку: make clean.
3. Соберите проект: make build-all.
4. Откройте build/site/en/<version>/index.html (или en/current/index.html) в браузере; для русской версии используйте build/site/ru/<version>/index.html.

## См. также

* [Архитектура](#system-architecture)
* [Установка и настройка](#installation)

# Руководство автора

В этом разделе описан процесс работы автора, отвечающего за создание и поддержку документации.

## Обзор

* Пишите материал в **AsciiDoc**.
* Проверяйте локально (**Vale** — стиль, **htmltest** — ссылки, предпросмотр сборки).
* Фиксируйте изменения и открывайте pull request.
* CI формирует **HTML** / **PDF** / **DOCX** и запускает QA-проверки.
* Мультиверсийный сайт по Git-тегам; для каждой версии доступны собственные загрузки в \_downloads.

## Структура проекта

docs/  
├── en/ # компонент для английской версии  
│ ├── antora.yml # метаданные компонента  
│ └── modules/ROOT/ # основной модуль Antora  
│ ├── pages/ # страницы AsciiDoc  
│ ├── assets/ # статика  
│ │ └── images/ # локальные изображения  
│ └── attachments/ # файлы для скачивания (идут в \_downloads)  
└── ru/ # компонент для русской версии  
 ├── antora.yml  
 └── modules/ROOT/  
 ├── pages/  
 ├── assets/  
 │ └── images/  
 └── attachments/

Каждый язык оформлен как отдельный компонент Antora; названия и версии могут отличаться.

## Подготовка окружения

1. Выполните [Installation & Setup](#installation).
2. Клонируйте репозиторий и установите пакеты Node.js:

* npm ci --no-audit --no-fund

1. Проверьте цепочку инструментов первичной сборкой:

* make build-all

## Режимы сборки

**Локальный** — по умолчанию. Собирается только текущая ветка **HEAD** или ветка, указанная в **BUILD\_REF**.

make build-all  
make build-all BUILD\_REF=my-feature

**Теги** — мультиверсионная сборка по всем Git-тегам.

make build-all BUILD\_SCOPE=tags

Переменные Make: **BUILD\_SCOPE** = **local** или **tags**, **BUILD\_REF** по умолчанию HEAD.

## Артефакты и размещение

### PDF

* build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* копируется в site/<locale>/<version>/\_downloads/adaptadocx-<locale>.pdf

### DOCX

* build/docx/<locale>/<version>/adaptadocx-<locale>.docx
* копируется в site/<locale>/<version>/\_downloads/adaptadocx-<locale>.docx

Ссылки **PDF** и **DOCX** в шапке указывают на \_downloads текущей версии.

## Ежедневный процесс

1. **Редактируйте** файлы .adoc в соответствующем языковом компоненте.
2. **Просматривайте** сайт локально:

* make build-html

1. **Проверяйте** стиль и ссылки:

* make test

1. **Формируйте все форматы** при необходимости пакета для ревью:

* make build-all

1. **Проверьте** артефакты в build/ и загрузки в site/<locale>/<version>/\_downloads.

## Проверка перед коммитом

Запускайте три основные проверки:

vale docs/  
make build-html  
htmltest -c .htmltest.yml build/site

## Процесс pull request

1. Создайте ветку фичи:

* git checkout -b feat/short-description

1. Внесите изменения и выполните локальную сборку:

* make build-all && make test

1. Закоммитьте, запушьте и откройте PR.

CI прикрепляет к запуску vale.xml и htmltest.log.

## Конвейеры CI

### Release (теги)

* триггер — пуш в тег;
* сборка в Docker с BUILD\_SCOPE=tags;
* проверка **htmltest** и **Vale**;
* загрузка собранного сайта и прод-деплой.

### QA checks (pull request в main)

* линт shell-скриптов, запуск **Vale**;
* сборка и **htmltest** для текущей ветки.

### Security audit (pull request в main, push в теги)

* неблокирующие проверки: **OSV-Scanner**, **Sandworm**, поиск запрещённых паттернов;
* краткая сводка публикуется в отчёте запуска.

## Процесс перевода

1. Напишите или обновите английскую страницу.
2. Скопируйте файл в зеркальный путь docs/ru/ и переведите.
3. Проверьте кросс-ссылки в обоих языках.
4. Запустите make build-html и убедитесь, что поиск работает.
5. Откройте pull request.

## Инструментарий

| Категория | Инструменты / Файлы |
| --- | --- |
| Редактирование | Редактор с поддержкой AsciiDoc |
| Валидация | **Vale**, **htmltest**, **Shellcheck** |
| Сборка | Makefile, Dockerfile |
| Конфигурация | antora-playbook-en.yml, antora-playbook-ru.yml, antora-assembler.yml, .vale.ini, config/default-theme.yml |
| CI | .github/workflows/release.yml, .github/workflows/qa-checks.yml, .github/workflows/security-audit.yml |

## См. также

* [Быстрый старт](#quick-start)
* [Установка и настройка](#installation)

# Руководство DocOps

## Архитектура

Adaptadocx — модульная система документации на базе **Antora**. Стек организован в пять слоёв:

1. Исходные файлы
2. Оркестрация сборки
3. Генерация форматов (HTML, PDF, DOCX)
4. Контроль качества
5. CI-процессы безопасности и упаковки

### Основные компоненты

| Слой | Каталог / Файл | Роль |
| --- | --- | --- |
| Контент | docs/en/, docs/ru/ | Языковые исходники AsciiDoc |
| Контент | docs/\*/modules/ROOT/pages | Отдельные страницы .adoc |
| Контент | docs/\*/antora.yml | Имя компонента, версия, навигация |
| Конфигурация | antora-playbook-en.yml, antora-playbook-ru.yml | Глобальная конфигурация сайта (источники, UI, вывод) |
| Конфигурация | antora-assembler.yml | Настройки ассемблера для экспортов PDF/DOCX |
| Конфигурация | config/ | Тема PDF и метаданные по локалям (meta-\*.yml) |
| UI | custom-ui/ | Переопределения CSS/JS и макетов (ссылки на загрузки ведут в \_downloads) |
| Сборка | Makefile, build.py | Основной и альтернативный оркестраторы |
| Контейнер | Dockerfile | Воспроизводимая цепочка инструментов для CI и локального запуска |
| CI | .github/workflows/\*.yml | Workflow для QA, security audit и релизов |

### Приоритеты конфигурации

| Приоритет | Файл | Область |
| --- | --- | --- |
| **1** | docs/\*/antora.yml | Компонент |
| **2** | antora-playbook-\*.yml | Весь сайт |
| **3** | Встроенные атрибуты | Отдельный документ |

### Конвейер сборки

1. **build-html** — Antora генерирует сайт в build/site/ и экспорт ассемблера в build/asm/. Объём сборки управляется переменными **BUILD\_SCOPE** (local по умолчанию или tags) и **BUILD\_REF** (по умолчанию HEAD).
2. **build-pdf** — использует build/asm/, создаёт версионированные PDF в build/pdf/<locale>/<version>/, затем копирует их в site/<locale>/<version>/\_downloads/.
3. **build-docx** — использует build/asm/, создаёт версионированные DOCX в build/docx/<locale>/<version>/, затем копирует их в site/<locale>/<version>/\_downloads/.
4. **make test** — запускает Vale, htmltest, Shellcheck.
5. **Упаковка** — make release архивирует результат (adaptadocx-docs-<version>.zip).

Каталог build/asm/ содержит промежуточные сборки от @antora/pdf-extension. Каталог build/site/ содержит итоговый HTML-сайт.

#### Поток HTML

AsciiDoc → Antora → UI bundle → Lunr index → HTML site

* **Поиск** — Lunr со стеммерами EN/RU.
* Диаграммы **SVG** рендерятся нативно в браузере.

#### Поток PDF

AsciiDoc → Asciidoctor PDF → theme → fonts → PDF

* **Тема** — config/default-theme.yml.
* **Шрифты** — DejaVu (латиница + кириллица).
* **SVG** — при наличии конвертируются rsvg-convert.

#### Поток DOCX

AsciiDoc → Pandoc → Lua filter → metadata → DOCX

**Команда (пример EN)**

pandoc --from asciidoc \  
 --to docx \  
 --lua-filter=docx/coverpage.lua \  
 --metadata-file=config/meta-en.yml \  
 -o build/docx/en/0.1.2/adaptadocx-en.docx \  
 docs/en/modules/ROOT/pages/\*.adoc

### Контроль качества

| Инструмент | Проверки | Вывод |
| --- | --- | --- |
| **Vale** | Стиль, орфография | vale.xml |
| **htmltest** | Целостность ссылок | htmltest.log |
| **Shellcheck** | Lint shell-скриптов | Консоль |

### Параметры объёма сборки

| Переменная | Назначение | Значение по умолчанию |
| --- | --- | --- |
| BUILD\_SCOPE | local — собирать только текущую ветку; tags — собирать все Git-теги | local |
| BUILD\_REF | Ветка/ссылка для локального режима (например, my-feature, HEAD) | HEAD |

### Контейнерный образ

* **База** — Node.js 20 + Ruby ≥ 2.7
* **Дополнительно** — Asciidoctor PDF, Pandoc, Vale, htmltest, Shellcheck, Graphviz, шрифты DejaVu

Образ собирается один раз и переиспользуется во всех задачах CI.

### CI-процессы

| Workflow | Задача | Триггер |
| --- | --- | --- |
| QA Checks | Vale, htmltest, Shellcheck | pull\_request (в main) |
| Security Audit | OSV-Scanner, Sandworm, проверка запрещённых паттернов | pull\_request → main, push → теги ('\*') |
| Release | Контейнерная сборка, мультиверсионные доки (BUILD\_SCOPE=tags), архив артефактов и деплой | push по тегам |

Релизный workflow публикует полный сайт и загружает архив с артефактами.

### См. также

* [Оркестрация сборки](#build-orchestration)
* [Генерация документов](#document-generation)
* [CI/CD процессы](#ci-cd-workflows)

## Установка и настройка

Adaptadocx можно запускать в Docker-контейнере (рекомендуется) или в предварительно настроенной локальной среде.

### Системные требования

#### Docker

* **Docker Engine 20.10+** (или Docker Desktop)
* ≈4 ГБ свободного дискового пространства
* Доступ в интернет для получения образа

#### Локальная установка

* **Node.js 20+** с npm
* **Python 3.11+** (опционально, для build.py)
* **Ruby ≥ 2.7** (требуется Asciidoctor PDF)
* **Graphviz** — рендер SVG и других диаграмм в PDF/DOCX
* **Vale** — проверка стиля и грамматики
* **htmltest** — проверка целостности ссылок
* **Shellcheck** — анализ shell-скриптов
* **Git**
* ≈2 ГБ свободного дискового пространства

### Установка в Docker (рекомендуется)

1. Клонируйте репозиторий

* git clone https://github.com/your-org/adaptadocx.git  
  cd adaptadocx

1. Постройте образ

* docker build -t adaptadocx .

1. Сгенерируйте полный набор документации

* # Только текущая ветка (по умолчанию)  
  docker run --rm -v "$(pwd)":/work adaptadocx make build-all  
    
  # Все теги (мультиверсийная сборка)  
  docker run --rm -v "$(pwd)":/work adaptadocx make build-all BUILD\_SCOPE=tags

### Локальная установка

1. Установите зависимости Node.js

* npm ci --no-audit --no-fund

1. Установите Ruby + Asciidoctor PDF

* # Debian / Ubuntu  
  sudo apt-get update  
  sudo apt-get install -y ruby ruby-dev  
  gem install asciidoctor-pdf

1. Установите инструменты QA и Graphviz

* # Vale  
  wget -qO- https://github.com/errata-ai/vale/releases/download/v2.29.4/vale\_2.29.4\_Linux\_64-bit.tar.gz \  
  | tar -xz && sudo mv vale /usr/local/bin/  
    
  # htmltest  
  wget -qO- https://github.com/wjdp/htmltest/releases/download/v0.17.0/htmltest\_0.17.0\_linux\_amd64.tar.gz \  
  | tar -xz && sudo mv htmltest /usr/local/bin/  
    
  # Shellcheck  
  sudo apt-get install -y shellcheck  
    
  # Graphviz  
  sudo apt-get install -y graphviz # Linux  
  brew install graphviz # macOS

1. Соберите все форматы

* # Только текущая ветка (по умолчанию)  
  make build-all  
    
  # Все теги (мультиверсийная сборка)  
  make build-all BUILD\_SCOPE=tags

1. Запустите проверки качества

* make test

### Расположение результатов

* **HTML** — build/site/<locale>/<version>/
* **PDF** — build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* **DOCX** — build/docx/<locale>/<version>/adaptadocx-<locale>.docx
* Публикация на сайте — site/<locale>/<version>/\_downloads/

### Диагностика

#### Docker

* **Ошибка прав при сборке** — убедитесь, что демон Docker запущен и пользователь входит в группу docker.
* **Ошибка записи в том** — проверьте права файловой системы на монтируемом пути.

#### Локальная среда

* **Неверная версия Node.js** — установите Node.js 20 через nvm.
* **Ошибка установки Ruby-gem** — проверьте наличие ruby-dev и компилятора C.
* **Vale / htmltest не найдены** — проверьте, что они доступны в $PATH.

#### Сборка

* **Проблемы со шрифтами в PDF** — установите шрифты DejaVu (sudo apt-get install -y fonts-dejavu).
* **SVG не попадают в PDF** — установите rsvg-convert (sudo apt-get install -y librsvg2-bin).
* **Сбой проверок QA** — изучите vale.xml, htmltest.log и вывод Shellcheck.

### См. также

* [Быстрый старт](#quick-start)
* [Архитектура](#system-architecture)

## Генерация документов

Adaptadocx формирует три типа артефактов из единого дерева AsciiDoc:

* **HTML** — двуязычный статический сайт с навигацией, поиском и скачиваемыми артефактами в DOCX и PDF.
* **PDF** — буклет для печати с настраиваемой темой.
* **DOCX** — редактируемый документ.

Все конвейеры используют общий контент и выход **assembler** Antora, а затем применяют формат-специфические шаги. Выходы версионируются по локали и по тегу.

### Конвейер HTML

**Схема потока HTML**

AsciiDoc → Antora → UI bundle → Lunr index → HTML site

#### Playbook Antora (EN)

Файл antora-playbook-en.yml

site:  
 title: Adaptadocx Documentation  
 start\_page: en::index.adoc  
 url: https://adaptadocx.netlify.app/en  
  
content:  
 branches: ~  
 tags: '\*'  
 sources:  
 - url: .  
 start\_path: docs/en  
  
urls:  
 html\_extension\_style: default  
  
ui:  
 bundle:  
 url: ./custom-ui/ui-bundle.zip  
 snapshot: true  
 supplemental\_files: ./custom-ui/supplemental\_ui  
  
output:  
 dir: ./build/site  
  
antora:  
 extensions:  
 - require: '@antora/pdf-extension'  
 configFile: ./antora-assembler.yml  
 - require: '@antora/lunr-extension'  
 languages: [en]

* **Search** — Lunr включается на уровне локали (см. languages).
* **UI bundle** — каталог custom-ui/ переопределяет макеты, CSS и JS. Ссылки в хедере указывают на версионированные загрузки в \_downloads.

#### Конфигурация assembler

Файл antora-assembler.yml

assembly:  
 root\_level: 0  
 section\_merge\_strategy: fuse  
 xml\_ids: true  
component\_version\_filter:  
 names: '\*\*'  
build:  
 dir: ./build/asm  
 keep\_source: true  
 command: 'true'  
 publish: false  
 qualify\_exports: true

**Assembler** складывает экспортированные деревья в build/asm/<locale>/<version>/\_exports/.

### Конвейер PDF

**Схема потока PDF**

AsciiDoc (из assembler) → Asciidoctor PDF → theme → fonts → PDF

Для каждой locale и version, обнаруженных в site/<locale>/<version>/:

1. Берём build/asm/<locale>/<version>/\_exports/index.adoc.
2. При наличии копируем изображения из build/asm/<locale>/<version>/\_images.
3. Рендерим через asciidoctor-pdf с revnumber=<version>.

Тема config/default-theme.yml:

extends: default  
font:  
 catalog:  
 DejaVu Sans:  
 normal: DejaVuSans.ttf  
 bold: DejaVuSans-Bold.ttf  
 italic: DejaVuSans-Oblique.ttf  
 bold\_italic: DejaVuSans-BoldOblique.ttf  
 DejaVu Sans Mono:  
 normal: DejaVuSansMono.ttf  
 bold: DejaVuSansMono-Bold.ttf  
page:  
 size: A4  
 margin: [2cm, 2cm, 2cm, 2cm]  
base:  
 font-family: DejaVu Sans  
 font-size: 11

#### Размещение выходов

* build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* копируется в site/<locale>/<version>/\_downloads/adaptadocx-<locale>.pdf

### Конвейер DOCX

**Схема потока DOCX**

AsciiDoc (из assembler) → Asciidoctor DocBook → Pandoc → Lua filters → DOCX

Для каждой locale и version:

1. Читаем build/asm/<locale>/<version>/\_exports/index.adoc.
2. Конвертируем в DocBook через asciidoctor -b docbook5.
3. Пайпим в pandoc с референсным DOCX и метаданными локали/версии.
4. При наличии rsvg-convert подключается фильтр конвертации SVG→PNG.

Пример (схема соответствует тому, что делает Makefile):

# Для каждой локали/версии:  
(cd "build/asm/<locale>/<version>/\_exports" && \  
 asciidoctor -b docbook5 \  
 -r extensions/collapsible\_tree\_processor.rb \  
 -a allow-uri-read -a revdate! -a revnumber! -a docdate! -a docdatetime! \  
 -o - index.adoc \  
| pandoc --from=docbook --to=docx \  
 --reference-doc=docx/reference.docx \  
 --metadata-file=config/meta-<locale>.yml \  
 --lua-filter=docx/coverpage.lua \  
 $( [ -x "$(command -v rsvg-convert)" ] && echo "--lua-filter=docx/svg2png.lua" ) \  
 -o "build/docx/<locale>/<version>/adaptadocx-<locale>.docx")

**Размещение выходов:** \* build/docx/<locale>/<version>/adaptadocx-<locale>.docx \* копируется в site/<locale>/<version>/\_downloads/adaptadocx-<locale>.docx

#### Фильтр титульной страницы

Файл docx/coverpage.lua

function Meta(meta)  
 meta.version = meta.version or os.getenv('VERSION') or 'dev'  
 return meta  
end

### Правила версионирования

* Локальная сборка по умолчанию формирует текущую ветку HEAD (или BUILD\_REF) — по одной версии на локаль.
* Релизная сборка использует все Git-теги (BUILD\_SCOPE=tags) — по нескольку версий на локаль.
* Ссылки из хедера ведут на \_downloads/adaptadocx-<locale>.(pdf|docx) внутри текущей версии.

### Диагностика

* **HTML** — битые ссылки → запустите make test и проверьте htmltest.log.
* **PDF** — отсутствует экспорт index.adoc → убедитесь, что есть build/asm/<locale>/<version>/\_exports/index.adoc для нужной версии.
* **DOCX** — ошибки парсинга Pandoc → проверьте Lua-фильтры (docx/coverpage.lua, docx/svg2png.lua) и DocBook-поток.
* **Шрифты** — установите пакет fonts-dejavu, если не хватает глифов.

### См. также

* [Архитектура](#system-architecture)
* [CI/CD процессы](#ci-cd-workflows)
* [Оркестрация сборки](#build-orchestration)

## Оркестрация сборки

### Обзор

Сборочный слой **умышленно дублирован**:

* **GNU Make** — основной вход, применяется CI и Docker
* **build.py** — эквивалент на Python 3.11+ для систем без **make**

Обе точки входа обслуживают общий конвейер:

* **Генерация** — HTML / PDF / DOCX
* **Контроль качества** — Vale, htmltest, Shellcheck
* **Упаковка** — ZIP-архив каждого билда
* **Контейнеризация** — полная воспроизводимость через docker build

Используйте либо Makefile, либо build.py — не смешивайте их в одном прогоне.

### Режимы сборки

**Локальный** — собирается только текущая ветка HEAD или ветка, заданная через BUILD\_REF.

make build-all  
make build-all BUILD\_REF=my-feature

**Теги** — мультиверсионная сборка по всем Git-тегам.

make build-all BUILD\_SCOPE=tags

Переменные Make: BUILD\_SCOPE = local или tags; BUILD\_REF по умолчанию HEAD.

### Артефакты и размещение

#### PDF

* build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* копируется в site/<locale>/<version>/\_downloads/adaptadocx-<locale>.pdf

#### DOCX

* build/docx/<locale>/<version>/adaptadocx-<locale>.docx
* копируется в site/<locale>/<version>/\_downloads/adaptadocx-<locale>.docx

### Цели Make

| Target | Назначение |
| --- | --- |
| make build-site | HTML + PDF + DOCX (все локали) |
| make build-html | только HTML |
| make build-pdf | только PDF |
| make build-docx | только DOCX |
| make test | Vale • htmltest • Shellcheck |
| make clean | удалить build/ |
| make release | ZIP + QA-проверка |
| make build-all | псевдоним → build-site |

### Точки входа Python

| Команда | Назначение |
| --- | --- |
| python3 build.py build-site | HTML + PDF + DOCX |
| python3 build.py build-html | Только HTML |
| python3 build.py build-pdf | Только PDF |
| python3 build.py build-docx | Только DOCX |
| python3 build.py test | Запуск тестов (Vale, htmltest при наличии сайта, Shellcheck) |
| python3 build.py clean | Удалить build/ |

**Параметры и значения по умолчанию**:

* --scope local|tags (по умолчанию local) **local** собирает только текущий ref; **tags** собирает все Git-теги.
* --ref <git-ref> (по умолчанию HEAD) Используется только при --scope local.
* Переменные окружения дублируют флаги: BUILD\_SCOPE, BUILD\_REF.

**Примеры**:

# Все теги (мультиверсионная сборка)  
python3 build.py --scope tags build-all  
  
# Локальная сборка для конкретной ветки  
python3 build.py --scope local --ref my-feature build-site

### Основные цели Make (сокращённо)

build-site: build-html build-pdf build-docx  
 @echo "[site] full build done"  
  
build-html:  
 @echo "[html] start"; \  
 for l in $(LOCALES); do \  
 echo " • $${l}"; \  
 pb="antora-playbook-$${l}.yml"; \  
 if [ "$(BUILD\_SCOPE)" = "tags" ]; then \  
 npx antora "$$pb"; \  
 else \  
 bak="$$pb.bak"; \  
 cp "$$pb" "$$bak"; \  
 tr -d '\r' < "$$pb" > "$$pb.unix" && mv "$$pb.unix" "$$pb"; \  
 sed -i "s/tags: '\\*'/tags: ~/" "$$pb"; \  
 sed -i "s/branches: ~$$/branches: $(BUILD\_REF)/" "$$pb"; \  
 npx antora "$$pb"; \  
 mv "$$bak" "$$pb"; \  
 fi; \  
 done  
 @echo "[html] done"  
  
build-pdf: build-html  
 @mkdir -p "$(PDF\_DIR)"; \  
 for l in $(LOCALES); do \  
 echo "[pdf] $$l"; \  
 for version\_dir in $(SITE\_DIR)/$$l/\*/; do \  
 if [ -d "$$version\_dir" ]; then \  
 version=$$(basename "$$version\_dir"); \  
 if [ "$(BUILD\_SCOPE)" != "tags" ] && [ "$$version" != "$(BUILD\_REF)" ] && [ "$$version" != "current" ] && [ "$$version" != "main" ]; then continue; fi; \  
 export\_file=""; \  
 for candidate in "$(ASM\_DIR)/$$l/$$version/\_exports/index.adoc" "$(ASM\_DIR)/$$l/\_exports/index.adoc" "$(ASM\_DIR)/\_exports/$$l/$$version/index.adoc" "$(ASM\_DIR)/\_exports/$$l/index.adoc"; do \  
 if [ -f "$$candidate" ]; then export\_file="$$candidate"; base=$$(dirname "$$(dirname "$$candidate")")); break; fi; \  
 done; \  
 [ -z "$$export\_file" ] && continue; \  
 img\_src="$$base/\_images"; img\_dst="$$(dirname "$$export\_file")/$$l/$$version/\_images"; \  
 [ -d "$$img\_src" ] && mkdir -p "$$img\_dst" && cp -r "$$img\_src"/\* "$$img\_dst"/ || true; \  
 outdir="$(PDF\_DIR)/$$l/$$version"; outfile="$$outdir/adaptadocx-$$l.pdf"; \  
 mkdir -p "$$outdir"; \  
 toc=$$( [ "$$l" = ru ] && echo '-a toc-title=Содержание' || echo '-a toc-title=Contents' ); \  
 asciidoctor-pdf $(ASCIIDOCTOR\_PDF\_OPTS) $$toc -a revnumber=$$version -o "$$outfile" "$$export\_file"; \  
 mkdir -p "$(SITE\_DIR)/$$l/$$version/\_downloads"; \  
 cp "$$outfile" "$(SITE\_DIR)/$$l/$$version/\_downloads/adaptadocx-$$l.pdf"; \  
 fi; \  
 done; \  
 done  
 @echo "[pdf] done"  
  
build-docx: build-html  
 @mkdir -p "$(DOCX\_DIR)"; \  
 for l in $(LOCALES); do \  
 echo "[docx] $$l"; \  
 for version\_dir in $(SITE\_DIR)/$$l/\*/; do \  
 if [ -d "$$version\_dir" ]; then \  
 version=$$(basename "$$version\_dir"); \  
 if [ "$(BUILD\_SCOPE)" != "tags" ] && [ "$$version" != "$(BUILD\_REF)" ] && [ "$$version" != "current" ] && [ "$$version" != "main" ]; then continue; fi; \  
 base="$(ASM\_DIR)/$$l/$$version"; \  
 img\_src="$$base/\_images"; img\_dst="$$base/\_exports/$$l/$$version/\_images"; \  
 [ -d "$$img\_src" ] && mkdir -p "$$img\_dst" && cp -r "$$img\_src"/\* "$$img\_dst"/ || true; \  
 outdir="$(DOCX\_DIR)/$$l/$$version"; outfile="$$outdir/adaptadocx-$$l.docx"; outfile\_abs="$(CURDIR)/$$outfile"; \  
 mkdir -p "$$outdir"; \  
 tmp\_meta="$(CURDIR)/$(DOCX\_DIR)/meta-$$l-$$version.yml"; \  
 sed "s/{page-version}/$$version/g" $(CURDIR)/config/meta-$$l.yml > "$$tmp\_meta"; \  
 ( cd "$$base/\_exports" && asciidoctor -b docbook5 -r $(CURDIR)/extensions/collapsible\_tree\_processor.rb -a allow-uri-read -a revdate! -a revnumber! -a docdate! -a docdatetime! -o - index.adoc | pandoc --from=docbook --to=docx --reference-doc=$(PANDOC\_REF) --metadata-file="$$tmp\_meta" $(SVG\_FILTER) --lua-filter=$(LUA\_COVER) -o "$$outfile\_abs" ); \  
 rm -f "$$tmp\_meta"; \  
 mkdir -p "$(SITE\_DIR)/$$l/$$version/\_downloads"; \  
 cp "$$outfile" "$(SITE\_DIR)/$$l/$$version/\_downloads/adaptadocx-$$l.docx"; \  
 fi; \  
 done; \  
 done  
 @echo "[docx] done"

#### Цели контроля качества

test:  
 @if [ -d "$(SITE\_DIR)" ]; then \  
 htmltest -c .htmltest.yml "$(SITE\_DIR)"; \  
 else \  
 echo "[test] Skipping htmltest - no site built"; \  
 fi  
 @vale --config=.vale.ini docs/  
 @find scripts -name '\*.sh' -print0 | xargs -0 -I{} bash -c 'tr -d "\r" < "{}" | shellcheck -'  
 @echo '[test] OK'

#### Служебные цели

clean:  
 -rm -rf build  
 @echo '[clean] build/ removed'  
  
release: build-site test  
 @cd build && zip -rq ../"$(RELEASE\_FILE)" .  
 @echo "[release] $(RELEASE\_FILE) created"

Где RELEASE\_FILE := adaptadocx-docs-$(VERSION).zip.

### Работа в контейнере

Образ Docker инкапсулирует весь тулчейн; типовые запуски:

# Сборка образа  
docker build -t adaptadocx:latest .  
  
# Полная сборка  
docker run --rm -v "$(pwd)":/work adaptadocx:latest make build-site  
  
# Только QA-проверки  
docker run --rm -v "$(pwd)":/work adaptadocx:latest make test  
  
# Интерактивная отладка  
docker run -it --rm -v "$(pwd)":/work adaptadocx:latest bash

### Конфигурационные переменные

|  |  |  |
| --- | --- | --- |
| Variable | Назначение | Default |
| LOCALES | Поддерживаемые языки | ru en |
| VERSION | Версия из Git/package.json | автоопределение |
| BUILD\_SCOPE | Режим сборки (local или tags) | local |
| BUILD\_REF | Ветка для локального режима | HEAD |
| SITE\_DIR | Каталог HTML-сайта | build/site |
| ASM\_DIR | Каталог сборки Antora | build/asm |
| PDF\_DIR | Каталог PDF-файлов | build/pdf |
| DOCX\_DIR | Каталог DOCX-файлов | build/docx |
| PANDOC\_REF | Референсный DOCX | docx/reference.docx |
| LUA\_COVER | Lua-фильтр обложки | docx/coverpage.lua |
| SVG\_FILTER | Lua-фильтр SVG→PNG | docx/svg2png.lua |
| RELEASE\_FILE | Имя архива релиза | adaptadocx-docs-$(VERSION).zip |

#### Определение версии

VERSION := $(shell git describe --tags --abbrev=0 2>/dev/null \  
 || node -p "require('./package.json').version")

### Отладка

* **Unknown target** — запускать make из корня репозитория
* **Stale artefacts** — make clean перед новой сборкой
* **CI mismatch** — версии инструментов в Docker должны совпадать с локальными

См. также: [CI/CD процессы](#ci-cd-workflows)

## Управление заголовками

Заголовки разрешаются по жёсткой трёхуровневой иерархии:

| Приоритет | Источник | Область | Переопределение |
| --- | --- | --- | --- |
| **1** | Файл компонента docs/\*/antora.yml | Имя и версия компонента | Переопределяет всё |
| **2** | Playbook antora-playbook-\*.yml | Брендинг сайта | Переопределяет документы |
| **3** | Атрибуты внутри .adoc | Отдельный файл | Самый низкий |

### Заголовки на уровне компонента

Файл docs/en/antora.yml

name: en  
title: Adaptadocx Documentation  
version: '1.0'  
display\_version: '1.0'  
  
asciidoc:  
 attributes:  
 component-title: '{title}'  
 component-version: '{version}'  
 document-title: '{component-title} {component-version}'

Русский компонент в docs/ru/antora.yml оформлен аналогично (локализованные строки, name: ru).

| Атрибут | Назначение |
| --- | --- |
| title | Подпись в левой навигации |
| version | Машиночитаемая версия |
| display\_version | Человекочитаемая версия |
| component-title | Переиспользуемый атрибут |
| document-title | Шаблон для производных заголовков |

### Заголовки на уровне playbook

Фрагмент antora-playbook-en.yml

site:  
 title: Adaptadocx Documentation  
 start\_page: en::index.adoc  
  
asciidoc:  
 attributes:  
 site-title: '{site.title}'  
 page-title-pattern: '{site-title} - {page-title}'

Русский playbook antora-playbook-ru.yml зеркален и использует локализованные значения.

| Атрибут | Назначение |
| --- | --- |
| site.title | Основной HTML <title> |
| site-title | Переиспользуемый атрибут |
| page-title-pattern | Глобальный шаблон HTML-<title> |

### Заголовки на уровне документа

Внутри любого .adoc-файла:

= Архитектура системы  
:navtitle: Архитектура

| Атрибут | Эффект | Можно переопределить |
| --- | --- | --- |
| = Title | H1 в отрендеренном контенте | Метаданные могут меняться |
| :navtitle: | Пункт меню | Может быть заменён навигацией компонента (nav.adoc) |

### Обработка при выводе

* **HTML** — <title> определяется по page-title-pattern → значения компонента → заголовок документа.
* **PDF** — заголовок/версия берутся из атрибутов компонента; Makefile передаёт -a revnumber=<version>.
* **DOCX** — метаданные загружаются из config/meta-en.yml / config/meta-ru.yml; для каждой версии Makefile создаёт временный файл build/docx/meta-<locale>-<version>.yml с подстановкой 0.1.0.

### Диагностика

* **Несогласованные заголовки** — проверьте title/version компонента и атрибуты playbook.
* **Неверный заголовок/версия в PDF** — проверьте использование -a revnumber в Makefile и настройки темы config/default-theme.yml.
* **Несоответствие меню и страницы** — корректируйте :navtitle: или nav.adoc компонента.

### См. также

* [Архитектура](#system-architecture)
* [CI/CD процессы](#ci-cd-workflows)
* [Оркестрация сборки](#build-orchestration)

## Управление версиями

Adaptadocx координирует публикации документации на разных языках с помощью **семантического версионирования**, метаданных компонентов Antora и CI-workflow, основанных на Git-тегах.

### Семантическое версионирование

Строка версии: **MAJOR.MINOR.PATCH**

| Сегмент | Пояснение |
| --- | --- |
| MAJOR | Несовместимые изменения |
| MINOR | Новые функции без ломки |
| PATCH | Исправления ошибок / только docs |

**Хронология примеров**

1.0.0 → первый релиз  
1.1.0 → новые возможности  
1.1.1 → фиксы / docs  
2.0.0 → несовместимое изменение

### Правила Git

**Тег и публикация**

git tag -a v1.2.0 -m "Release 1.2.0"  
git push origin --tags

Типы веток

| Ветка | Назначение | Шаблон |
| --- | --- | --- |
| main | Стабильный код | main |
| Feature | Новая функциональность | feature/\* |
| Release | Подготовка релиза (редко) | release/vX.Y.Z |
| Hot-fix | Срочный патч | hotfix/vX.Y.Z |

### Версии компонентов Antora

У каждой локали свой antora.yml. Для **сборок по тегам** Antora подставляет версию из Git-тега. Для **сборок по веткам** контент публикуется под псевдоверсией current (или именем ветки). Плейсхолдеры в репозитории не требуются.

**Английский**

name: adaptadocx  
version: 'current' # сборка по тегам переопределит автоматически  
title: Adaptadocx Documentation

**Русский**

name: adaptadocx  
version: 'current' # сборка по тегам переопределит автоматически  
title: Документация Adaptadocx

### Логика определения версии

Используется fallback-стратегия, чтобы и релизы по тегам, и сборки по веткам разрешались корректно.

**Структура каталогов**

build/asm/  
├── <locale>/  
│ ├── 1.2.0/ ← релиз по тегу v1.2.0  
│ └── current/ ← сборка по ветке без тега

1. Ищем …/<version>/.
2. Если нет, используем …/current/.

### CI и объём сборки

* **Локальные / PR-сборки** — по умолчанию BUILD\_SCOPE=local, собирается только текущая ветка (BUILD\_REF=HEAD).
* **Релизные сборки** — BUILD\_SCOPE=tags, Antora строит все версии по Git-тегам.

**Пример (release workflow)**

- name: Build docs in container  
 run: |  
 docker run --rm -v "${{ github.workspace }}:/work" adaptadocx:latest \  
 bash -lc 'npm ci --no-audit --prefer-offline && make clean && make build-all BUILD\_SCOPE=tags'

### Размещение артефактов

Версионирование кодируется **каталогами**, имена файлов стабильные.

| Формат | Путь вывода |
| --- | --- |
| PDF (EN) | build/pdf/en/<version>/adaptadocx-en.pdf → копируется в site/en/<version>/\_downloads/adaptadocx-en.pdf |
| PDF (RU) | build/pdf/ru/<version>/adaptadocx-ru.pdf → копируется в site/ru/<version>/\_downloads/adaptadocx-ru.pdf |
| DOCX (EN) | build/docx/en/<version>/adaptadocx-en.docx → копируется в site/en/<version>/\_downloads/adaptadocx-en.docx |
| DOCX (RU) | build/docx/ru/<version>/adaptadocx-ru.docx → копируется в site/ru/<version>/\_downloads/adaptadocx-ru.docx |

Ссылки в шапке UI указывают на \_downloads **текущей версии**, например:

* '\_downloads/adaptadocx-en.pdf'
* '\_downloads/adaptadocx-en.docx'
* '\_downloads/adaptadocx-ru.pdf'
* '\_downloads/adaptadocx-ru.docx'

### Чек-лист релиза

1. Обновите version в package.json при необходимости.
2. При изменении ветки или start\_path — проверьте playbook’и.
3. Соберите обе локали, проверьте индекс поиска.
4. Поставьте тег и запушьте (vX.Y.Z).
5. Убедитесь, что артефакты EN и RU лежат в site/<locale>/<version>/\_downloads/ и build/(pdf|docx)/<locale>/<version>/.

### См. также

* [Архитектура](#system-architecture)
* [CI/CD процессы](#ci-cd-workflows)
* [Оркестрация сборки](#build-orchestration)

## CI/CD процессы

Adaptadocx автоматизирует линтинг, QA, проверки безопасности и пакетные сборки с помощью **GitHub Actions**. Артефакты поставляются как ZIP, а также как версионированные загрузки внутри сайта.

### Матрица процессов

| Процесс | Триггер | Задания |
| --- | --- | --- |
| QA Checks | pull\_request → **main** | **Shellcheck** · **Vale** · **htmltest** (параллельно), сборка в Docker |
| Security Audit | pull\_request → **main**, push → теги ('\*') | **OSV-Scanner** · **Sandworm** · banned-pattern scan (неблокирующий) |
| Release | push → теги ('\*') | Docker build → make build-all BUILD\_SCOPE=tags → **htmltest** + **Vale** → ZIP + upload artefacts |
| Deploy | после **Release** на тег | Загрузка артефакта сайта → деплой в Netlify --prod |

### QA Checks

Файл: /.github/workflows/qa-checks.yml

* Задания: **shellcheck**, **vale**, **htmltest**.
* Триггер: pull\_request в main.
* Сайт для **htmltest** собирается в том же Docker-образе, что и релиз.

Шаги сборки в задании **htmltest**:

- name: Build docs image  
 run: docker build -t adaptadocx:latest .  
  
- name: Build docs in container  
 run: |  
 docker run --rm \  
 -v "${{ github.workspace }}:/work" \  
 adaptadocx:latest \  
 bash -lc 'npm ci --no-audit --prefer-offline && make clean && make build-all'

Загрузка логов выполняется всегда (пример):

- name: Upload htmltest log  
 if: always()  
 uses: actions/upload-artifact@v4  
 with:  
 name: htmltest-log  
 path: htmltest.log

### Security Audit

Файл: /.github/workflows/security-audit.yml

* Триггер: pull\_request → main и push в теги ('\*').
* Шаги: **OSV-Scanner**, **Sandworm audit**, проверка запрещённых паттернов; затем краткая сводка в $GITHUB\_STEP\_SUMMARY.
* Поведение: все проверки запускаются с continue-on-error: true, поэтому аудит сообщает о находках, но **не блокирует** PR.
* Выходные файлы:
  + reports/osv.json — отчёт OSV (пропускается, если lock-файлы отсутствуют)
  + reports/sandworm.json — отчёт @sandworm/audit
  + reports/banned-patterns-report.txt — результат пользовательского grep-gate

Ключевые фрагменты:

**OSV-Scanner**

- name: OSV scan  
 id: osv  
 continue-on-error: true  
 shell: bash  
 run: |  
 files=$(git ls-files | grep -E 'package-lock\.json$|pnpm-lock\.yaml$|yarn\.lock$' || true)  
 if [[ -z "$files" ]]; then  
 echo "scanned=false" >> "$GITHUB\_OUTPUT"  
 echo "No lockfiles → skipping OSV"  
 exit 0  
 fi  
 docker run --rm -v "$PWD:/src" -w /src ghcr.io/google/osv-scanner:latest \  
 --format json --output /src/reports/osv.json $files || true  
 echo "scanned=true" >> "$GITHUB\_OUTPUT"

**Sandworm audit**

- name: Sandworm audit  
 id: sandworm  
 continue-on-error: true  
 run: npx -y @sandworm/audit@latest --json > reports/sandworm.json

**Бан-паттерны**

- name: Banned patterns  
 id: banned  
 continue-on-error: true  
 run: node scripts/scan-banned-patterns.cjs

**Сводка**

- name: Summarise results  
 if: always()  
 shell: bash  
 run: |  
 echo '### Security audit summary' >> "$GITHUB\_STEP\_SUMMARY"  
 hits=$(grep -c '^BANNED' reports/banned-patterns-report.txt 2>/dev/null || echo 0)  
 echo "\*\*Banned-pattern hits:\*\* $hits" >> "$GITHUB\_STEP\_SUMMARY"  
  
 if [[ "${{ steps.osv.outputs.scanned }}" == "true" ]]; then  
 echo 'OSV scan ✔' >> "$GITHUB\_STEP\_SUMMARY"  
 else  
 echo 'OSV scan ⏭ (skipped)' >> "$GITHUB\_STEP\_SUMMARY"  
 fi  
  
 [[ -f reports/sandworm.json ]] \  
 && echo 'Sandworm scan ✔' >> "$GITHUB\_STEP\_SUMMARY" \  
 || echo 'Sandworm scan ✖' >> "$GITHUB\_STEP\_SUMMARY"

### Release

Файл: /.github/workflows/release.yml

Два задания: **build** и **deploy**.

#### Build

* Собирает Docker-образ.
* Выполняет мультиверсийную сборку по тегам через BUILD\_SCOPE=tags.
* Запускает **htmltest** и **Vale** внутри контейнера.
* Загружает логи и артефакты, архивирует build/ в docs-${{ github.sha }}.zip.

Фрагмент:

- name: Build docs image  
 run: docker build -t adaptadocx:latest .  
  
- name: Build docs in container  
 run: |  
 docker run --rm \  
 -v "${{ github.workspace }}:/work" \  
 adaptadocx:latest \  
 bash -lc 'npm ci --no-audit --prefer-offline && make clean && make build-all BUILD\_SCOPE=tags'

#### Deploy

Запускается только для пуша в тег. Публикует ранее выгруженный сайт в Netlify.

deploy:  
 needs: build  
 runs-on: ubuntu-latest  
 if: github.event\_name == 'push' && github.ref && startsWith(github.ref, 'refs/tags/')  
 steps:  
 - name: Download built site  
 uses: actions/download-artifact@v4  
 with:  
 name: built-site  
 path: site  
  
 - name: Deploy to Netlify  
 run: |  
 npx netlify-cli deploy \  
 --dir=site \  
 --site="${{ secrets.NETLIFY\_SITE\_ID }}" \  
 --auth="${{ secrets.NETLIFY\_AUTH\_TOKEN }}" \  
 --prod

### Что именно собирается

* В QA сборке сайт формируется для текущей ветки (режим Make по умолчанию **BUILD\_SCOPE=local**) и проверяется **htmltest** на build/site.
* В релизной сборке формируются **все теги** (**BUILD\_SCOPE=tags**), чтобы для каждой версии были загрузки:
  + site/<locale>/<version>/\_downloads/
  + сопутствующие артефакты лежат в build/pdf/<locale>/<version>/ и build/docx/<locale>/<version>/.

### Отладка

* Воспроизвести шаг локально:
* docker build -t adaptadocx:latest .  
  docker run -it --rm -v "$PWD":/work adaptadocx:latest bash
* Проверить граф зависимостей Make: make -d build-all
* Убедиться, что раннер видит историю и теги (actions/checkout@v4 с fetch-depth: 0 и git fetch --tags origin)

# FAQ

В этом разделе собраны ответы на популярные вопросы по установке, настройке, использованию и диагностике Adaptadocx.

## Установка и настройка

**Каковы минимальные системные требования?**

Для работы Adaptadocx требуются:

* **Node.js 18+** с менеджером пакетов npm.
* **Ruby ≥ 2.7** — используется Asciidoctor PDF.
* **4 ГБ свободного диска** — для исходников и артефактов сборки.
* **Git** — получение репозитория и контроль версий.

При установке через Docker хост-требования сокращаются до **Docker Engine 20.10+** и тех же 4 ГБ места.

**Докер или локальная установка?**

В большинстве случаев предпочтителен Docker, так как он обеспечивает:

* **Единообразие** — одинаковая версия инструментов на любой платформе.
* **Простоту** — контейнер содержит все зависимости.
* **Изоляцию** — не затрагивает существующие утилиты на хосте.
* **Надёжность** — образ протестирован и гарантированно собирается.

Локальную установку используйте только если инструменты действительно нужны на хосте или требуется интеграция с существующей средой.

**Как проверить корректность установки?**

Соберите минимальный HTML-пакет и убедитесь в наличии вывода.

# Вариант с Docker  
docker run --rm -v "$(pwd)":/work adaptadocx:latest make build-html  
  
# Вариант без Docker  
make build-html

При успехе HTML-артефакты появятся в:

* build/site/en/<version>/
* build/site/ru/<version>/

Также может присутствовать псевдоверсия current/ в зависимости от конфигурации Antora.

## Система сборки

**Откуда берётся ошибка "No rule to make target"?**

Makefile не нашёл указанную цель. Возможные причины:

* **Неверный каталог** — запускайте make из корня проекта.
* **Отсутствует Makefile** — убедитесь, что файл есть и читается.
* **Опечатка** — выведите список доступных целей:
* grep -E '^[A-Za-z0-9\_-]+:' Makefile | cut -d: -f1 | sort -u
* **Права доступа** — проверьте, что Makefile читаем текущим пользователем.

## Форматы вывода

**Где сохраняются сгенерированные PDF и DOCX?**

Каждая сборка формирует **версионированные** артефакты по локали:

* **PDF** — build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* **DOCX** — build/docx/<locale>/<version>/adaptadocx-<locale>.docx

Копии публикуются в загрузках сайта:

* site/<locale>/<version>/\_downloads/adaptadocx-<locale>.pdf
* site/<locale>/<version>/\_downloads/adaptadocx-<locale>.docx

**Почему кириллица искажена в PDF?**

Убедитесь, что шрифты DejaVu доступны системе, где запускается Asciidoctor PDF.

# Debian / Ubuntu  
sudo apt-get install -y fonts-dejavu fonts-dejavu-extra  
  
# Проверка установки  
fc-list | grep -i dejavu  
  
# Убедитесь, что тема использует DejaVu  
grep -i dejavu config/default-theme.yml

**Как изменить оформление PDF?**

Редактируйте файл темы config/default-theme.yml. Поддерживаются все ключи Asciidoctor PDF (наборы шрифтов, размеры заголовков, отступы и т. д.).

**Могу ли я настроить обложку DOCX?**

Да. Измените Lua-фильтр docx/coverpage.lua, например чтобы всегда заполнялись заголовок и версия:

function Meta(meta)  
 meta.title = meta.title or pandoc.MetaString(os.getenv('DOC\_TITLE') or 'Adaptadocx Documentation')  
 meta.version = meta.version or os.getenv('VERSION') or 'dev'  
 return meta  
end

**Как исправить битые ссылки в HTML?**

Запустите проверку ссылок и исправьте неверные адреса.

make build-html  
htmltest -c .htmltest.yml build/site  
  
# Просмотр лога  
cat htmltest.log

## Контроль качества

**Как расширить словарь Vale?**

Добавьте разрешённые термины в .vale/vocab/<DictionaryName>/accept.txt.

# .vale/vocab/Adaptadocx/accept.txt  
Adaptadocx  
AsciiDoc  
Antora  
DOCX

**Почему htmltest даёт ложные срабатывания?**

Настройте исключения в .htmltest.yml.

IgnoreURLs:  
 - "http://localhost"  
 - "https://example.com"  
  
HTTPStatusIgnore:  
 - 429 # Ограничение по запросам  
 - 503 # Сервис недоступен

## Многоязычная документация

**Можно ли добавить язык помимо EN и RU?**

Да, следуйте существующей схеме компонентов:

1. Создайте каталог docs/<locale>/.
2. Добавьте файл компонента docs/<locale>/antora.yml.
3. Добавьте playbook antora-playbook-<locale>.yml.
4. Создайте метаданные config/meta-<locale>.yml.
5. Включите стемминг поиска для нового языка в конфигурации Lunr.

## CI/CD и деплой

**Почему сборка в GitHub Actions падает, а локально проходит?**

Частые причины:

* **Разные версии инструментов** — в CI могут быть более новые или старые версии.
* **Чувствительность к регистру** — различия Windows vs. Linux файловых систем.
* **Ограничения ресурсов** — память или таймауты runner’а.
* **Отсутствие секретов** — переменные не заданы в настройках репозитория.

Включите подробный вывод (set -x, --debug и т. п.) в шагах workflow, чтобы локализовать проблему.

## Миграция и обновления

**Как перенести существующую документацию на Adaptadocx?**

Рекомендуемый поэтапный процесс:

1. **Конверсия контента** — переведите материалы в AsciiDoc.
2. **Организация структуры** — разнесите файлы по компонентам и модулям Antora.
3. **Конфигурация** — создайте файлы antora.yml и playbook’и.
4. **Обновление ссылок** — замените жёсткие пути на xref.
5. **Тестирование** — запустите make build-all && make test.

## См. также

* [Архитектура](#system-architecture)
* [CI/CD процессы](#ci-cd-workflows)
* [Оркестрация сборки](#build-orchestration)