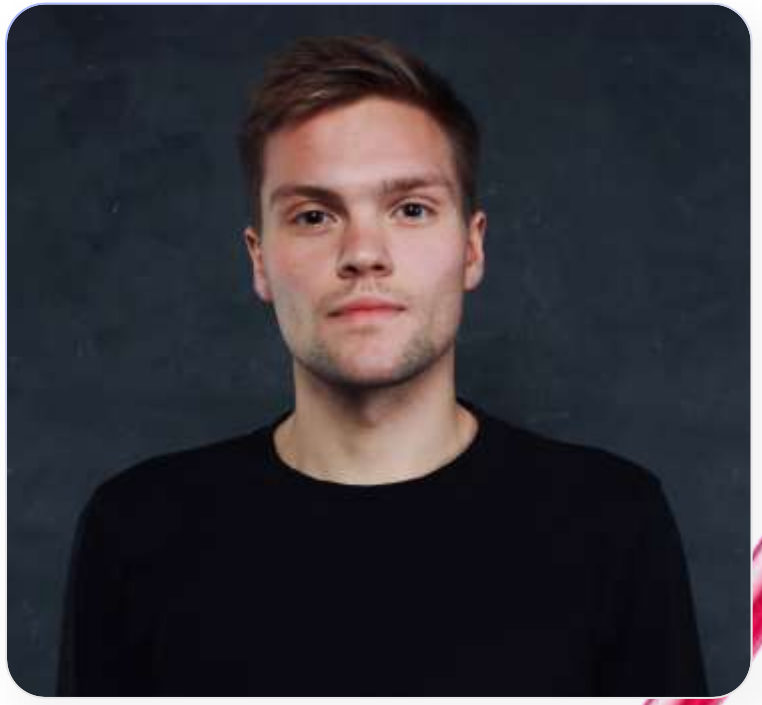
Практические шаблоны и лучшие практики для разработки Python-библиотек

Про себя

- Наша команда разрабатывает **удобные ETL-инструменты** в виде Python-библиотек, которые упрощают пользовательский опыт работы с open-source инструментами.

- Наши инструменты **повышают производительность разработчиков**, упрощают интеграцию данных и снижают затраты на поддержку процессов.



Python-разработчик МТС,
DataOPS Platform
команда ETL

Шаги для создания, публикации и установки пакета

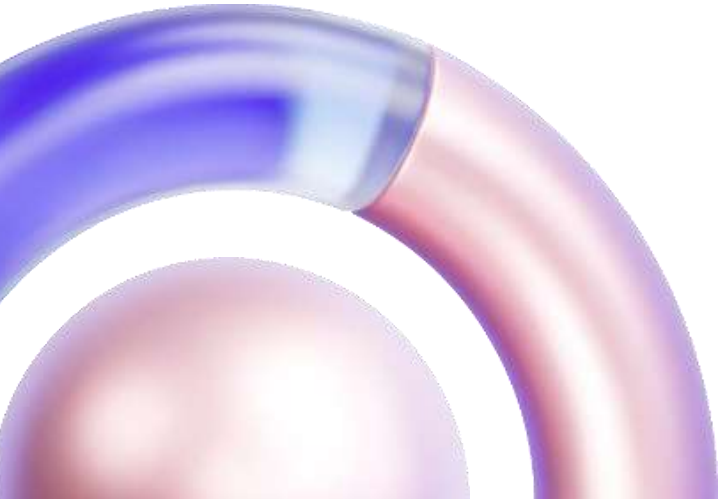
Шаг #1
Создание структуры проекта

Шаг #2
Создание pyproject.toml или setup.py

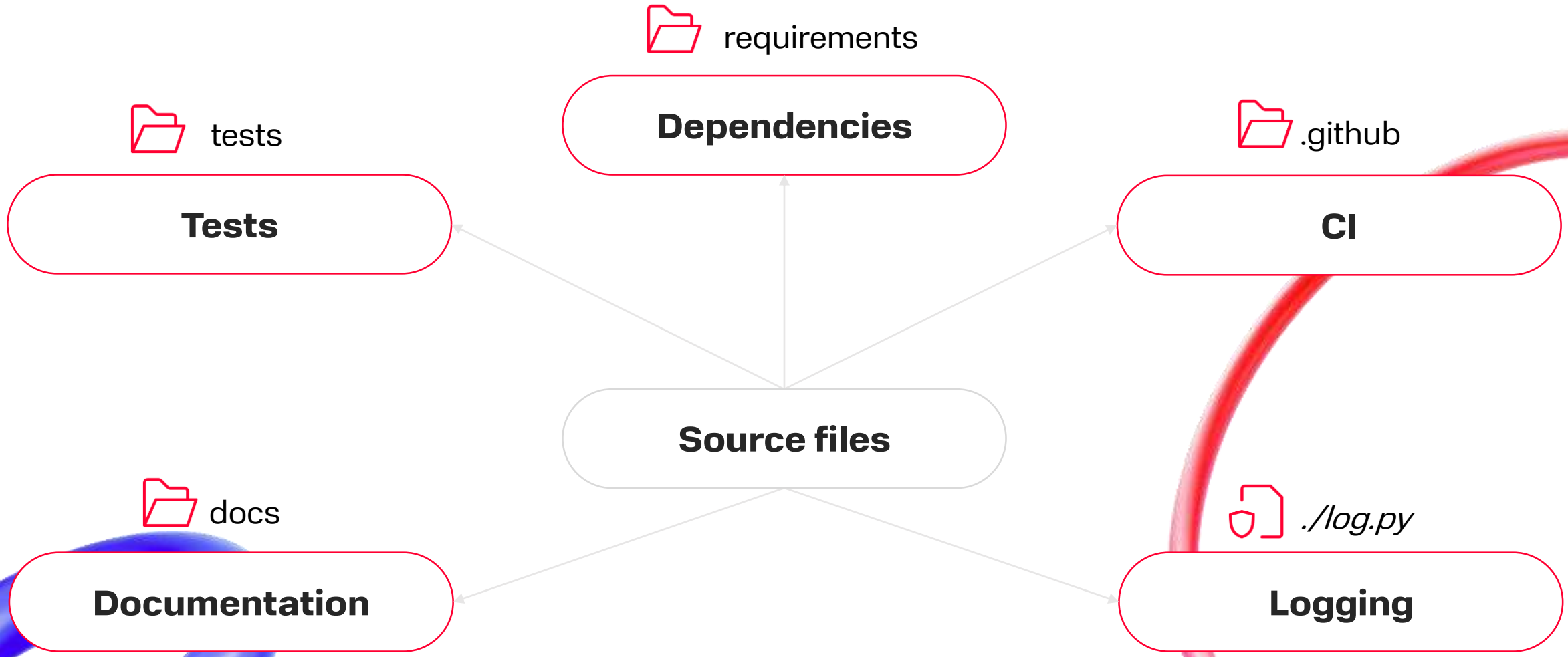
Шаг #3
Сборка пакета

Шаг #4
Публикация пакета на PyPI

Шаг #5
Установка пакета



Разделы


requirements
Dependencies
tests
Tests
.github
CI
Source files
docs
Documentation
./log.py
Logging

Tests

tests_unit

fixtures

tests_integration

.coveragerc

Tests

pyproject.toml (poetry)

```
[tool.poetry.group.test.dependencies]
pytest = "^7.4.4"
coverage = [
  {version = "^7.2.7", python = "<3.8"},
  {version = "^7.4.0", python = ">=3.8"},
]

# другие зависимости
```

requirements-test.txt (setuptools)

```
coverage
pytest<8
# другие зависимости
```

Tests

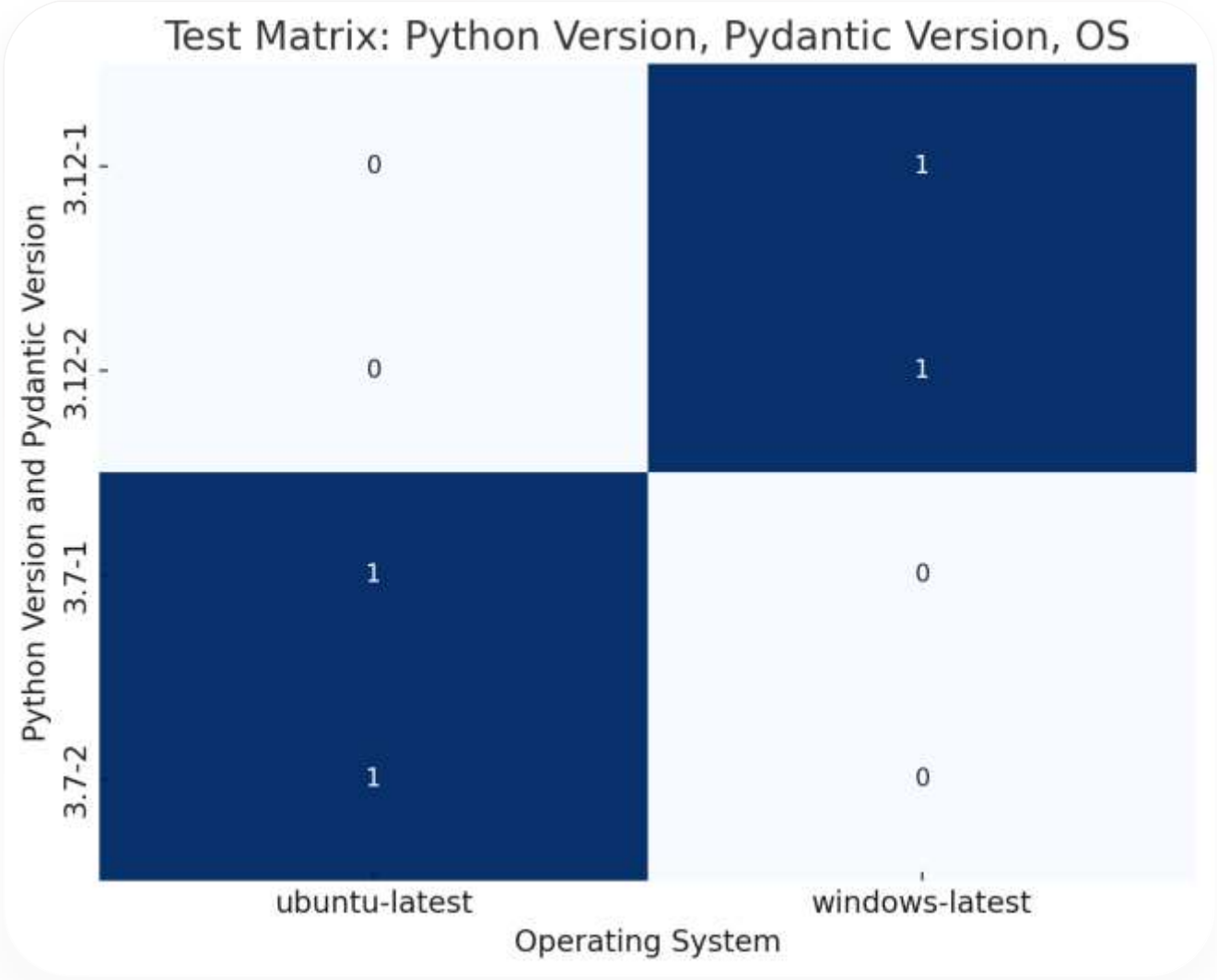
Unit Tests
test modules, test functions

~100%

Integration Tests
test interactions between components

Library Tests
test installation in editable mode, plugins import

Tests


Test Matrix: Python Version, Pydantic Version, OS
Python Version and Pydantic Version
3.12-1
3.12-2
3.7-1
3.7-2
0
1
0
1
1
0
1
0
ubuntu-latest
windows-latest
Operating System

Codecov -> Coveralls

.github/workflows/codeql.yml

Codecov Github Action

required

1. Клонировать репозиторий
2. Настроить Python
3. Установить зависимости
4. Скачать все отчеты по покрытию
5. Сгенерировать отчеты по покрытию
6. Проверить покрытие

Codecov -> Coveralls

Tests

- **Структура директорий для тестов**. Организуйте тесты в папки tests_unit для модульных тестов, tests_integration для интеграционных тестов и fixtures для общих тестовых данных.

Tests

- **Структура директорий для тестов**. Организуйте тесты в папки tests_unit для модульных тестов, tests_integration для интеграционных тестов и fixtures для общих тестовых данных.

- **Динамическая тестовая матрица**. Автоматически генерирует матрицу тестирования на основе обнаруженных изменений, обеспечивая полное покрытие тестами для всех поддерживаемых конфигураций.

Tests

- **Структура директорий для тестов**. Организуйте тесты в папки tests_unit для модульных тестов, tests_integration для интеграционных тестов и fixtures для общих тестовых данных.

- **Динамическая тестовая матрица**. Автоматически генерирует матрицу тестирования на основе обнаруженных изменений, обеспечивая полное покрытие тестами для всех поддерживаемых конфигураций.

- **Отдельные зависимости для тестов**. Поддерживайте отдельную группу зависимостей для тестирования (например, тестовую группу в pyproject.toml или requirements-test.txt).

Dependencies

Dependencies

pyproject.toml

poetry

- [tool.poetry.dependencies]
- [tool.poetry.group.test.dependencies] *(pytest, coverage, etc.)*
- [tool.poetry.group.dev.dependencies] *(sphinx, autodoc-pydantic, etc.)*
- [tool.poetry.group.docs.dependencies] *(black, mypy, wps, pre-commit , etc.)*

Dependencies

```
> pip install lib[backend,ldap]
```

poetry

pyproject.toml

```
[tool.poetry.extras]

backend = [...]
postgres = [...]
ldap = [...]
client-sync = [...]
```

setuptools

setup.py

```
setup(
extras_require={
        ”backend": requirements_backend,
        ”postgres": requirements_postgres,
        ”ldap": requirements_ldap,
        ”client-sync": requirements_client_sync
    },
)
```

Dependencies

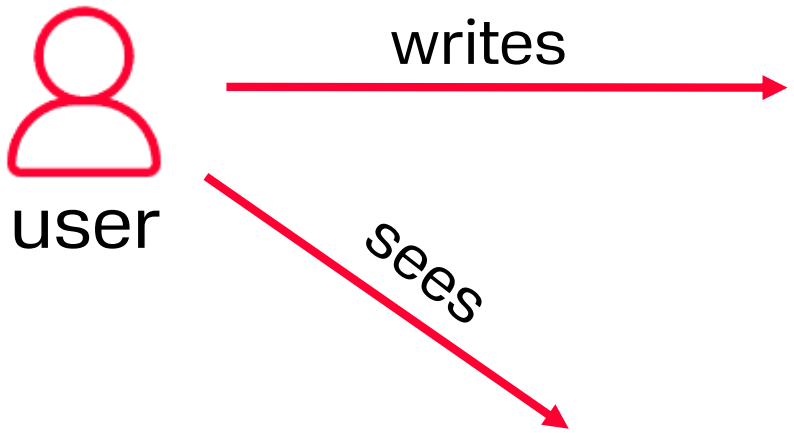
• **Разделение зависимостей на группы (тестовые, для разработки, для документации, для исходного кода)** повышает ясность, улучшает эффективность CI/CD и позволяет разработчикам устанавливать только то, что необходимо для их задач.

Dependencies

• **Разделение зависимостей на группы (тестовые, для разработки, для документации, для исходного кода)** повышает ясность, улучшает эффективность CI/CD и позволяет разработчикам устанавливать только то, что необходимо для их задач.

• **Использование дополнительных зависимостей (extras) для независимых частей библиотеки** обеспечивает модульную установку, уменьшает конфликты зависимостей, минимизирует занимаемую память и предоставляет пользователям гибкость в установке только тех компонентов, которые им нужны.

Logging

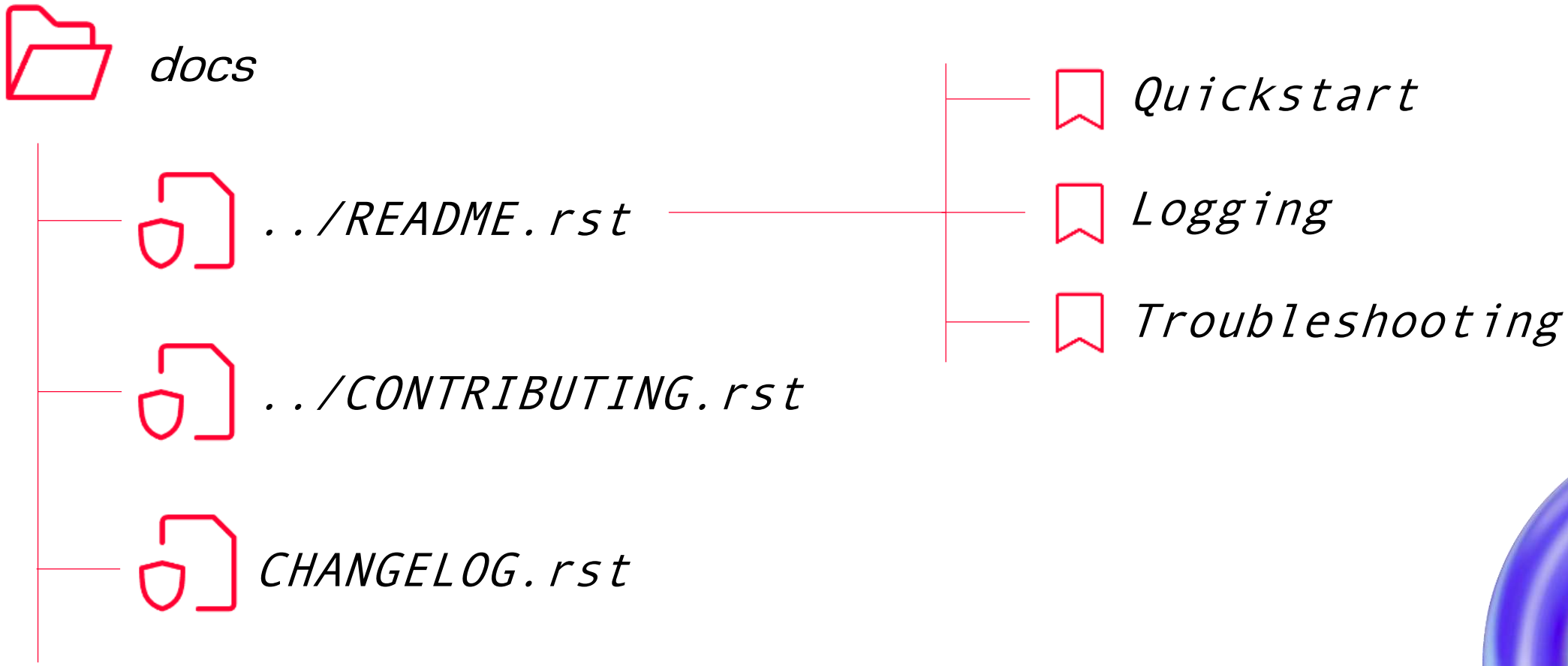



```python
from onetl.log import setup_logging

setup_logging(level="DEBUG",
              enable_clients=True)
```

```
2024-04-12 10:12:12,181 [INFO    ] MainThread: |IncrementalStrategy| Fetched HWM:
2024-04-12 10:12:12,182 [INFO    ] MainThread:         hwm = ColumnDateTimeHWM(
2024-04-12 10:12:12,182 [INFO    ] MainThread:             name = “some_name”,
2024-04-12 10:12:12,182 [INFO    ] MainThread:             entity = ‘table’,
2024-04-12 10:12:12,182 [INFO    ] MainThread:             expression = 'updated_at',
2024-04-12 10:12:12,184 [INFO    ] MainThread:         )
```

Documentation



Documentation

CONTRIBUTING.rst

- Fork and clone the repository.
- Set up a virtual environment and install dependencies.
- Running Tests
- Building Documentation
- Creating Pull Requests

Documentation

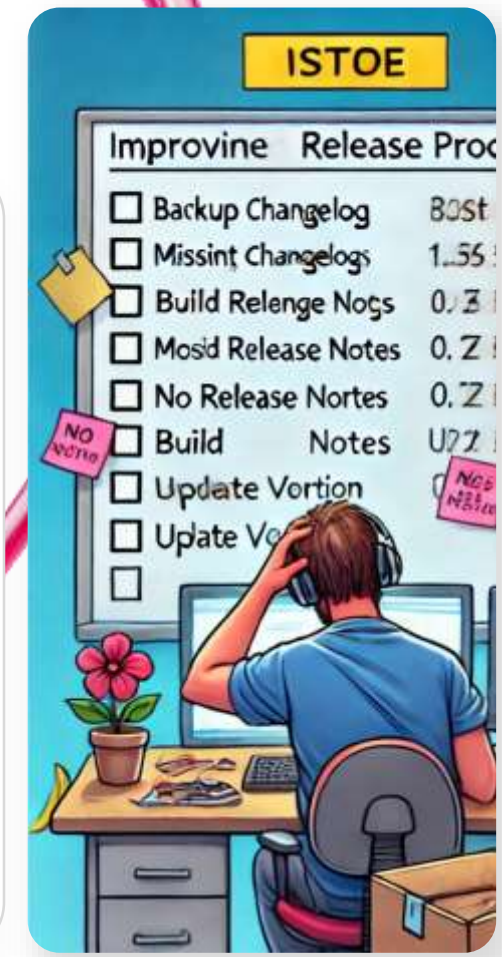
CONTRIBUTING.rst

Release Process

Step N. Add git tag to the latest commit in ``master`` branch

```
.. code:: bash

    git tag "$VERSION"
    git push origin "$VERSION"
```



Documentation

CHANGELOG.rst <PR_numer>.<change_type>.rst

feature

bugfix

misc

improvement

doc

dependency

Documentation

- **Добавьте руководство по внесению изменений (Contributing Guide)**. Обеспечьте возможность участия сообщества в open-source проектах, предоставляя чёткие руководства, что улучшит сотрудничество.

Documentation

- **Добавьте руководство по внесению изменений (Contributing Guide)**. Обеспечьте возможность участия сообщества в open-source проектах, предоставляя чёткие руководства, что улучшит сотрудничество.

- **Реализуйте процесс выпуска релизов**. Стандартизируйте и оптимизируйте процесс релизов, чтобы избежать ошибок, обеспечить согласованность и поддерживать плавный рабочий процесс для всех разработчиков.

Documentation

- **Добавьте руководство по внесению изменений (Contributing Guide)**. Обеспечьте возможность участия сообщества в open-source проектах, предоставляя чёткие руководства, что улучшит сотрудничество.

- **Реализуйте процесс выпуска релизов**. Стандартизируйте и оптимизируйте процесс релизов, чтобы избежать ошибок, обеспечить согласованность и поддерживать плавный рабочий процесс для всех разработчиков.

- **Предоставьте журнал изменений (Changelog)**. Отслеживайте развитие вашего проекта, что облегчит поиск, когда были добавлены или удалены функции, и поможет пользователям быть в курсе обновлений.

CI

codeql.yml

pre-commit black mypy flake8

```
pyupgrade.....................................................Passed
autoflake.....................................................Passed
black.........................................................Passed
blacken-docs..................................................Passed
bandit........................................................Passed
isort.........................................................Passed
Validate docker compose files.................................Passed
Pretty format YAML............................................Passed
codespell.....................................................Passed
poetry check..................................................Passed
poetry lock...................................................Passed
flake8........................................................Passed
mypy..........................................................Passed
```

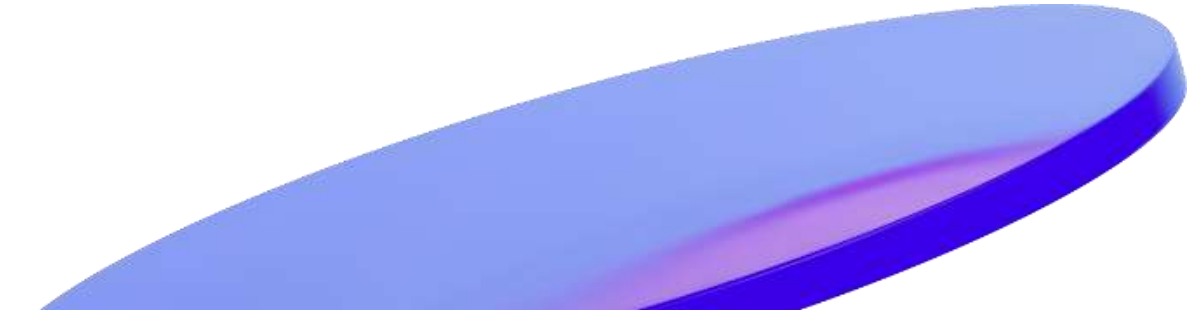
CI

`dev-release.yml`

При любых пушах в https://test.pypi.org/

`release.yml`

При релизных пушах в
(master, develop) to https://pypi.org/



PR

changelog.yml

✓ Сообщение коммита и заголовок PR являются информативными.

✓ Изменение должно быть минимальным.

✓ Существуют модульные и интеграционные тесты для внесённых изменений.

✓ Тесты проходят в CI, и уровень покрытия тестами не снижается.

test.yml

✓ Документация отражает внесённые изменения, где это применимо.

✓ Добавлен файл в docs/changelog/next_release/<pull request or issue id>.<change type>.rst, описывающий изменения (см. CONTRIBUTING.rst для подробностей).

✓ Мой PR готов к обзору.

codeql.yml

CI

- **Добавьте задачу CodeQL**. Обеспечьте качество и безопасность кода, автоматически запуская линтеры, форматтеры и другие инструменты статического анализа.

CI

- **Добавьте задачу CodeQL**. Обеспечьте качество и безопасность кода, автоматически запуская линтеры, форматтеры и другие инструменты статического анализа.

- **Запускайте pre-commit хуки перед отправкой**. Улавливайте потенциальные проблемы на ранней стадии, обеспечивая форматирования до того, как изменения будут отправлены в репозиторий.

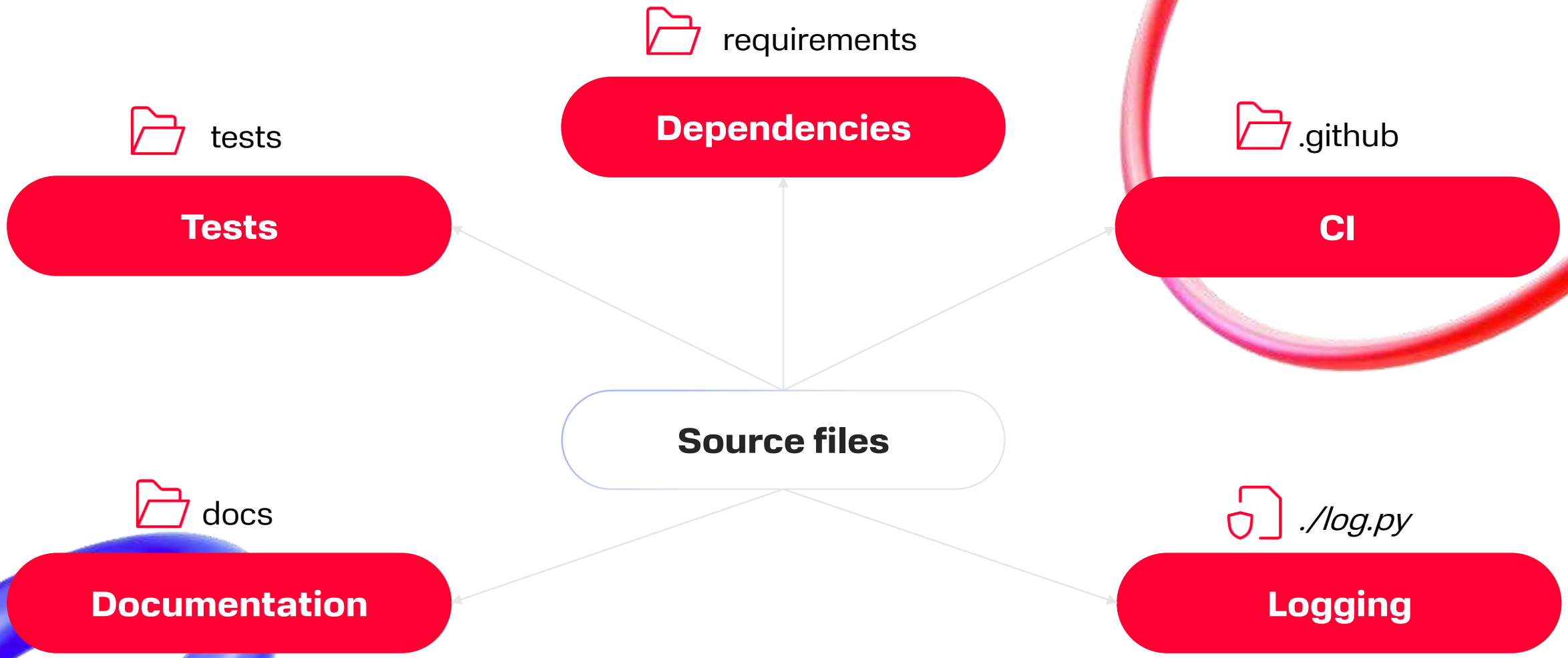
CI

- **Добавьте задачу CodeQL**. Обеспечьте качество и безопасность кода, автоматически запуская линтеры, форматтеры и другие инструменты статического анализа.

- **Запускайте pre-commit хуки перед отправкой**. Улавливайте потенциальные проблемы на ранней стадии, обеспечивая форматирования до того, как изменения будут отправлены в репозиторий.

- **Добавьте обязательные правила для создания PR.** Убедитесь в новых изменениях с помощью CI jobs и не тратьте время коллег.

Разделы


requirements
Dependencies
tests
Tests
.github
CI
Source files
docs
Documentation
./log.py
Logging

Результаты

- Сокращение времени между релизами
- Сокращение времени реализации новых инструментов
- Улучшение взаимодействия в команде и скорость принятия pull requests
- Уменьшение количества багов и проблем с зависимостями
- Быстрое вникание новых членов команд в проект

Наши проекты

onETL

Horizon

SyncMaster

ВОПРОСЫ