Чеклист по оформлению репозитория

19 июля 2023 г.

Лицензия	
Лицензия правильно применяется к репозиторию	
Используемые третьесторонние компоненты и материалы совместимы с лицензией	
Настроенный CI	
Модульные тесты в CI	
Линтер в CI	
В репозитории нет результатов сборки, настроен .gitignore	
В репозитории нет секретной информации (паролей, ключей и т.п.)	
Различные сторонние анализаторы (если уместно)	
README.md, плашки CI и анализаторов	
README.md, общее описание проекта	
README.md, пример использования	
README.md, инструкция по сборке и запуску	
README.md, как помочь проекту (если уместно)	
Код соответствует принятому в сообществе стилю кодирования	
Имеется техническая документация (в README.md или на вики)	
В коде достаточно комментариев	
Комментарии к коммитам адекватны, коммиты показывают историю проекта	

Пояснения

- Отдавайте предпочтение разрешающим лицензиям. Код мы рекомендуем лицензировать под Apache License 2.0, MIT License, BSD 3-Clause License.
- Каждая лицензия имеет требования к тому, как её правильно применить к файлам в репозитории. Например, Apache License 2.0 позволяет себя применять пофайлово, для чего требует включения в лицензируемые файлы стандартного заголовка. Также

распространено использование файла LICENSE в корне репозитория и ссылка на него в заголовке каждого файла. Поищите для своей лицензии, как её правильно применять.

- Если используете чужую интеллектуальную собственность, найдите на неё лицензию и проверьте, что вы действительно выполняете её требования (например, проект, лицензированный под Apache License 2.0 *не может* использовать код, лицензированный под GPL v2). Если чужой материал не имеет лицензии (например, просто картинка из интернета или кусок кода со Stack Overflow), использовать его *нельзя*.

- Если вы используете GitHub, Continuous Integration-систему удобнее всего настраивать на GitHub Actions, однако вполне допустимы и сторонние системы, такие как AppVeyor, CircleCI. Если вы используете компилируемые языки, CI-система должна проверять собирабельность кода в каждой ветке репозитория и при пуллреквесте. Если интерпретируемые, проверять качество кода и работоспособность.

- В проекте должны быть модульные тесты (за редкими исключениями, где они неприменимы или бессмысленны), и модульные тесты должны запускаться в CI.

- Должен быть настроен линтер, следящий за качеством кода, и также запускаться в CI. Если линтер выдаёт ошибки, сборка должна не проходить.

  - Например, для F# это Fantomas или FSharpLint, для Python — pylint и т.д.
  - Запуск линтера может быть отдельной задачей в CI, чтобы не гонять его по нескольку раз в разных конфигурациях сборки.
  - Имеет смысл сделать запуск линтера локальным pre-commit hook в git, чтобы некорректный код даже не позволяли закоммитить.

- На GitHub файл .gitignore можно выбрать при создании репозитория, но также часто требуется ручная модификация. Должно быть так, чтобы все файлы, которые .gitignore позволяет закоммитить, реально нужно было коммитить. *В репозитории не должно быть результатов сборки,* (то есть папок bin, obj, pycache и т.п.), в идеале не должно быть бинарных файлов вовсе (только если очень надо и вы реально знаете, что делаете).

- Разумеется, в репозитории (включая историю коммитов) не должно быть ничего, что вы не хотели бы публиковать (например, ключей авторизации от сообществ ВКонтакте).

- Используйте сторонние анализаторы для слежения за качеством кода: например, CodeCov для анализа тестового покрытия, CodeFactor или Codacy как продвинутый статический анализатор. Чем больше инструментов следят за тем, что всё хорошо, тем лучше.

- Добавьте в README.md плашки CI и анализаторов (штучки, на которых написано «CI passing» или что-то такое). В документации конкретной CI-системы или анализатора обычно легко найти, как добавить плашку в Markdown. Это поможет посетителям сразу посмотреть статус кода.

- Напишите в README.md пару абзацев текста, про что вообще проект. Помните, что код вы пишете не только для себя, в ваш репозиторий придут люди, которые вообще не имеют идей, о чём это.

- Опишите типичный пример использования, если уместно, с картинками или gif-ками. Включая информацию, откуда брать датасеты, куда подкладывать конфигурацию и т.п., чтобы любой пользователь мог с чистого листа запустить проект и понять, что у него получилось.

- Опишите также действия по сборке и внешние зависимости (версию используемых SDK и т.п.). Это всё есть в CI, но в README это всё должно быть в удобной человекочитаемой форме и заодно приводить к развёртыванию окружения, пригодного для работы над проектом (тогда как сборка в CI может быть весьма хитрой, использовать несколько Docker-образов и т.п.).

- Если проект предполагает возможность стороннего участия (то есть имеет хоть один шанс стать знаменитым), опишите, как сторонний человек может вам помочь:

  - куда и как писать баги;
  - как связаться с разработчиками;
  - как контрибьютить;
  - где посмотреть техническую документацию и найти первый вводный баг, который можно поправить.

- Проверьте, что код в репозитории адекватно оформлен. Если на Python, то PEP-8, если на C++, то в соответствии с Core Guidelines и т.п. — у каждого языка и даже у некоторых фреймворков есть свой стиль кодирования, проверьте, что код его уважает. Если в проекте используется свой стиль кодирования, он должен быть явно задокументирован и весь код должен ему соответствовать.

- Где-то должно быть некое техническое описание проекта — из каких компонентов он состоит, кто за что отвечает. В идеале — полноценная архитектурная документация в виде страниц на вики, с UML-диаграммами, но если сил нет, можно ограничиться разделом в README, где кратко словами всё описать.

- В коде должны быть комментарии (в принятом для языка формате — DocString, Doxygen, Javadoc, XML Documentation и т.п.), хотя бы у ключевых классов/интерфейсов/модулей, кратко описывающие, что вообще делает класс. В идеале — для всего, что public, с документированием предположений о входных данных, инвариантов, бросаемых исключений и свойств потокобезопасности (reentrant, thread-safe и т.п.), но насколько сил хватит.

- Комментарии к коммитам пост-фактум исправить тяжело, поэтому за ними надо следить изначально. Рекомендуется следовать соглашению Conventional Commits. Если у вас в репозитории больше пяти маловменяемых комментариев (типа «fix») подряд, лучше либо сделайте squash и склейте коммиты в один, либо измените всю историю через git rebase -i. Коммиты не должны быть сделаны в последний день, а должны показывать, как шла работа, от создания пустого проекта до последнего релиза. Фразы

вида «я тут локально разрабатывал, потом выложил, как получилось что-то разумное» очень сильно огорчают комиссию. Могут помочь инструменты типа Mergeable, Mergify.

Полезные ссылки для любопытных

- Правильное именование коммитов: https://www.conventionalcommits.org/en/v1.0.0/.
- Исправление комментариев к коммитам, редактирование истории (не делайте так без крайней нужды): https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History.
- Хорошее описание .gitignore: https://www.atlassian.com/ru/git/tutorials/saving-changes/gitignore.
- Модели работы с ветками:
  - https://www.atlassian.com/ru/git/tutorials/comparing-workflows;
  - https://www.gitkraken.com/learn/git/best-practices/git-branch-strategy;
  - https://www.endoflineblog.com/oneflow-a-git-branching-model-and-workflow.
- Зачем нужен README.md и как его писать: https://bulldogjob.com/readme/how-to-write-a-good-readme-for-your-github-project.
- Шаблон README с плашками и другими хорошими вещами: https://github.com/othneildrew/Best-README-Template.
- Подборка примеров хороших README: https://github.com/matiassingers/awesome-readme.
- Генератор README: https://readme.so/ru.
- Гайд по лицензиям от GitHub: https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository.
- Слайды лекции про авторские права в IT от Я.А. Кириленко: https://docs.google.com/presentation/d/1-xMvM_EyouDM9slBpQHR3h7MRgYrMQCKZ2dxYeFKh20.
- Может быть полезен dependabot (https://docs.github.com/en/code-security/dependabot), для автоматического обновления внешних зависимостей в проекте.