Программирование: как выбрать формат кода в 2026 году — стандарты, линтеры и ИИ

17.11.2026 · PDF-версия статьи с сайта cms.dolinais.ru

Почему формат кода снова стал важным

В 2026 году код всё чаще пишут не только люди, но и ИИ-агенты. По данным опросов разработчиков, большинство команд уже используют генерацию кода, а значит, единый стиль нужен не для эстетики, а для предсказуемости: чем меньше вариаций, тем проще ревью, автоматизация и обучение моделей на вашей кодовой базе. В этой статье разберём, как выбрать формат кода и настроить проверки, не превращая процесс в бюрократию.

Программирование — процесс создания и модификации компьютерных программ; по выражению Никлауса Вирта, «программы = алгоритмы + структуры данных». Формат кода — это соглашение о том, как эти алгоритмы и структуры записываются в тексте: отступы, кавычки, длина строк, порядок импортов. Сам по себе формат не влияет на работу программы, но влияет на скорость её понимания и изменения.

Три уровня договорённостей

Чтобы не утонуть в спорах, разделите правила на три уровня.

- **Форматирование** — чисто механические правила: отступы, пробелы, переносы. Их должен выполнять инструмент, а не человек.
- **Линтинг** — правила, которые ловят потенциальные ошибки и антипаттерны: неиспользуемые переменные, опасные конструкции, нарушение соглашений об именовании.
- **Архитектурные соглашения** — то, что не проверить автоматически: слои, границы модулей, работа с ошибками. Здесь нужны код-ревью и документация.

Начните с первого уровня: если форматирование автоматизировано, на ревью не будет споров о пробелах. Затем добавьте линтер с минимальным набором правил, которые действительно ловят ошибки в вашем стеке. Архитектурные соглашения фиксируйте в документе и обсуждайте на ревью.

Шаг 1. Выберите инструмент форматирования

В 2026 году почти для каждого языка есть устоявшийся форматтер. Для JavaScript и TypeScript это Prettier, для Python — Black или Ruff format, для Go — gofmt, для Rust — rustfmt, для Java — google-java-format. Главное правило: инструмент должен быть один на проект и запускаться автоматически.

1. Установите форматтер и зафиксируйте его версию в зависимостях проекта.
2. Добавьте конфигурационный файл в репозиторий, чтобы настройки были одинаковыми у всех.
3. Настройте запуск форматтера в pre-commit-хуке или в CI. Идеально — и там, и там: хук ускоряет локальную работу, CI гарантирует, что в основную ветку не попадёт неотформатированный код.
4. Прогоните форматтер по всей кодовой базе одним коммитом, чтобы не смешивать изменения стиля с логикой.

Если в проекте уже есть свой стиль, не ломайте его ради моды. Договоритесь, что новый код пишется по новым правилам, а старый форматируется постепенно, по мере правок.

Шаг 2. Подключите линтер с умом

Линтер — инструмент статического анализа, который ищет проблемы в коде без его запуска. В 2026 году популярны ESLint (JS/TS), Ruff (Python), golangci-lint (Go), Clippy (Rust). Ошибка новичков — включить все правила сразу. Это приводит к сотням предупреждений, которые никто не читает.

Поступите иначе:

- Начните с рекомендованного набора правил для вашего фреймворка.
- Добавляйте правила постепенно, по одному-два в спринт, и сразу исправляйте нарушения.
- Разделите правила на «ошибки» (ломают сборку) и «предупреждения» (не ломают, но видны в отчёте).
- Для унаследованного кода используйте baseline-файл: он фиксирует текущие нарушения и не даёт им расти.

  Правило, которое не соблюдается, хуже отсутствующего: оно создаёт иллюзию контроля и приучает игнорировать отчёты.

Шаг 3. Согласуйте стиль с ИИ-агентами

ИИ-агенты генерируют код на основе контекста, который вы им даёте. Если в проекте есть конфигурация форматтера и линтера, агент часто следует ей автоматически — особенно если вы передаёте ему правила в промпте или в файле инструкций. Практические шаги:

1. Держите конфигурации форматтера и линтера в корне репозитория — их видят и люди, и агенты.
2. Добавьте короткий файл с правилами для ИИ: какие команды запускать, какие ограничения соблюдать. Например, «перед коммитом запусти formatter и linter».
3. Включите в CI проверку, что сгенерированный код отформатирован. Это дисциплинирует и людей, и агентов.
4. Не позволяйте агенту менять конфигурацию линтера без ревью: иначе он может «подкрутить» правила под свой код.

Важно: автоматическое форматирование не заменяет ревью. Оно снимает механическую работу, но не проверяет логику, безопасность и соответствие архитектуре.

Шаг 4. Встройте проверки в процесс

Чтобы формат не стал предметом разовых договорённостей, встройте его в ежедневный процесс.

- **Локально:** pre-commit-хук запускает форматтер и линтер для изменённых файлов. Это быстро и не мешает работе.
- **В CI:** отдельная джоба проверяет форматирование и линт. Она должна быть быстрой и падать с понятным сообщением.
- **В ревью:** проверяйте только то, что не автоматизировано. Если ревьюер пишет «поставь пробел», значит, процесс настроен плохо.

Для больших монорепозиториев используйте инкрементальные проверки: линтуйте только изменённые пакеты. Это экономит время и снижает стоимость CI.

Распространённые ошибки

- **Слишком много правил сразу.** Команда перестаёт читать отчёты, и линтер превращается в формальность.
- **Разные настройки у разработчиков.** Если конфигурация не в репозитории, у каждого будет свой стиль.
- **Форматирование в одном коммите с логикой.** Это затрудняет ревью и поиск ошибок.
- **Игнорирование унаследованного кода.** Без baseline-файла старые нарушения будут всплывать при каждой правке.
- **Доверие агенту без проверки.** Сгенерированный код может быть отформатирован, но содержать логические ошибки.

Что в итоге

В 2026 году выбор формата кода — это не вкусовщина, а инженерная задача. Настройте один форматтер, постепенно добавляйте правила линтера, зафиксируйте конфигурации в репозитории и встройте проверки в CI. Такой подход снижает трение в команде, упрощает ревью и делает код понятным и людям, и ИИ-агентам. А главное — освобождает время для задач, которые действительно требуют человеческого суждения.

Источник: https://cms.dolinais.ru/programmirovanie-kak-vybrat-format-koda-v-2026-godu-standarty-lintery-i-ii