



TrebleSnake 12 июля в 23:06

Как следует писать комментарии к коммитам

<https://chris.beams.io/posts/git-commit/>

Системы управления версиями, Программирование, Git

Перевод

	КОММЕНТАРИЙ	ДАТА
○	НАПИСАЛ ГЛАВНЫЙ ЦИКЛ И УПРАВЛЕНИЕ ТАЙМЕРОМ	14 ЧАСОВ НАЗАД
○	ДОБАВИЛ ПАРСИНГ ФАЙЛА НАСТРОЕК	9 ЧАСОВ НАЗАД
○	РАЗНЫЕ БАГФИКСЫ	5 ЧАСОВ НАЗАД
○	ТАМ ДОБАВИЛ, ТУТ ИСПРАВИЛ	4 ЧАСА НАЗАД
○	БОЛЬШЕ КОДА	4 ЧАСА НАЗАД
○	ВОТ ТЕБЕ ЕЩЕ КОД	4 ЧАСА НАЗАД
○	АААААААА	3 ЧАСА НАЗАД
○	ФВЛАОЫДЛВАОЫВЛДАО	3 ЧАСА НАЗАД
○	МОИ ПАЛЬЦЫ НАБИРАЮТ СЛОВА	2 ЧАСА НАЗАД
○	ПАААААЛЦЫЫЫЫ	2 ЧАСА НАЗАД

ЧЕМ ДОЛЬШЕ ТЯНЕТСЯ ПРОЕКТ, ТЕМ МЕНЕЕ ИНФОРМАТИВНЫ
СООБЩЕНИЯ МОИХ GIT-КОММИТОВ.

Предисловие от переводчика

На протяжении многих лет разработки ПО, будучи участником многих команд, работая с разными хорошими и опытными людьми, я часто наблюдал (да и чего греха таить, до определенного момента — создавал) одну и ту же проблему — тотальный бардак в репозитории. Каждый писал комментарии к коммитам в своем стиле (и хорошо, если постоянно в одном); половина комментариев была бесполезна (из разряда "э́мост"), половина оставшейся половины — едва понятна.

И вот в один прекрасный момент я увидел данную статью, до перевода которой у меня наконец дошли руки. Всего 7 простых и коротких правил, и — о чудо — смотреть на историю коммитов стало не только полезно, но и приятно. Ничего революционного, все довольно очевидно, но сформулировано и резюмировано просто отлично.

Хочу заметить, что это статья 2014 года. Какие-то не сильно релевантные вещи, упоминаемые автором, могли потерять актуальность, но суть статьи — ничуть.

Введение: чем важны хорошие комментарии

Если вы заглянете в случайный Git-репозиторий, то, скорее всего, обнаружите, что в истории коммитов там в какой-то мере бардак. Например, взгляните на [эти перлы](#) из того времени, когда я начинал коммитить в репозиторий Spring:

```
$ git log --oneline -5 --author cbeams --before "Fri Mar 26 2009"

e5f4b49 Re-adding ConfigurationPostProcessorTests after its brief removal in r814. @Ignore-ing the testCglibClass
2db0f12 fixed two build-breaking issues: + reverted ClassMetadataReadingVisitor to revision 794 + eliminated Co
147709f Tweaks to package-info.java files
22b25e0 Consolidated Util and MutableAnnotationUtils classes into existing AsmUtils
7f96f57 polishing
```

Жуть. Сравните с этими, [более свежими](#), коммитами в том же репозитории:

```
$ git log --oneline -5 --author pwebb --before "Sat Aug 30 2014"

5ba3db6 Fix failing CompositePropertySourceTests
84564a0 Rework @PropertySource early parsing logic
e142fd1 Add tests for ImportSelector meta-data
887815f Update docbook dependency and generate epub
ac8326d Polish mockito usage
```

Какой вариант вы бы предпочли?

Первые разнятся по длине и стилю, вторые — лаконичные и однородные. Первое получается само собой, второе пишется осознанно.

И хотя история коммитов многих репозиторий выглядит подобно первому варианту, есть исключения. Отличные примеры — [ядро Linux](#) и [св Git](#). Посмотрите также на [Spring Boot](#) или любой другой репозиторий, которым занимается [Tim Pope](#).

Участники этих проектов знают, что хорошо написанный комментарий к коммиту — это лучший способ рассказать о *контексте* сделанных изменений другим разработчикам (а также самим себе — в будущем). Различия в ревизиях показывают, *что* изменилось, но только комментарий может внятно объяснить — *почему*. Peter Hutterer хорошо [сказал об этом](#):

Восстановление обстоятельств написания кода — расточительная трата времени. Мы не можем полностью ее избежать, поэтому наши усилия должны быть сосредоточены на минимизации этих затрат. Именно для этого и нужны комментарии к коммитам. Следовательно, они показывают, хорошо ли программист работает в команде.

Если вы не особо задумывались о том, каким должен быть первоклассный комментарий коммита — вероятно, вы не слишком часто использовали **git log** и похожие инструменты. Тут замкнутый круг: так как история коммитов неструктурированная и разнородная, ей не пользуются и не уделяют внимания. А из-за того, что ей не пользуются и не уделяют внимания, она остается неструктурированной и разнородной.

Но хорошо оформленная история репозитория — вещь прекрасная и полезная. Оживают команды **git blame**, **revert**, **rebase**, **log**, **shortlog** и прочие. Появляется смысл просматривать чужие коммиты и пулл-реквесты, и, внезапно, при этом теперь не требуется помощь их авторов. Разобраться, почему что-то случилось [в коде] месяцы или годы назад становится не только возможно, но и удобно.

Долгосрочный успех проекта зависит (помимо всего прочего) от того, насколько его удобно поддерживать, а история коммитов — один из самых мощных инструментов мейнтенера. Стоит потратить время на то, чтобы научиться сохранять в ней порядок. Сначала это может причинять некоторые неудобства, но потом войдет в привычку и, в конце концов, станет источником гордости и продуктивной работы всех участников.

Эта статья затрагивает лишь самую базовую составляющую ведения добротной истории, а именно — как следует писать комментарий к отдельному коммиту. Есть и другие важные вещи, например, объединение коммитов, которые здесь не рассматриваются.

В большинстве языков программирования есть хорошо описанные общепринятые соглашения, формирующие характерный стиль [написания кода], как то — имена переменных, правила форматирования и так далее. Конечно, есть разные версии таких соглашений, но большинство разработчиков придерживаются мнения, что выбрать один вариант и следовать ему — гораздо лучше, чем неразбериха, когда каждый пишет в своем стиле.

Подход команды к описанию коммитов должен быть точно таким же. Чтобы история репозитория была полезной, команда должна прийти к соглашению как минимум по трем следующим пунктам.

Стиль. Синтаксис разметки, отступы, переносы строк, грамматика, заглавные буквы, пунктуация. Всегда проверяйте орфографию и пишите как можно проще. В результате вы получите удивительно цельную историю коммитов, которую не только приятно читать, но которую действительно *будут* регулярно читать.

Содержимое. Какая именно информация должна (если вообще должна) содержаться в теле комментария? А чего там *не должно* быть?

Метаданные. Как следует ссылаться на ID задач, номера пулл-реквестов и т.д.?

К счастью, уже существуют соглашения о написании содержательного комментария. На самом деле, они частично вытекают из того, как работают некоторые команды Git. Вам не нужно изобретать велосипед. Просто следуйте [семи правилам](#) ниже — и будете на шаг ближе к истории коммитов, достойной профессионала.

Семь правил классного комментария к коммиту

Помните: [Все это уже было сказано раньше](#).

1. [Отделяйте заголовок от тела пустой строкой](#)
2. [Ограничивайте заголовок 50 символами](#)
3. [Пишите заголовок с заглавной буквы](#)
4. [Не ставьте точку в конце заголовка](#)

5. Используйте повелительное наклонение в заголовке
6. Переходите на следующую строку в теле на 72 символах
7. В теле отвечайте на вопросы *что* и *почему*, а не *как*

Например:

Резюмируйте изменения в 50 символах или меньше

Тут объясните их более детально, если необходимо. Выполняйте переносы строк примерно на 72 символах. В некоторых ситуациях первая строка комментария считается его заголовком, а все остальное – телом. Крайне важно отделять одно от другого пустой строкой (если у сообщения вообще есть тело, конечно); различные инструменты типа ``log``, ``shortlog`` и ``rebase`` не поймут вас, если заголовок и тело не будут отделены.

Объясните здесь, какую проблему решает этот коммит. Уделите больше внимания тому, почему вы внесли эти изменения, а не тому, как именно вы это сделали (это объяснит за вас код). Есть ли сайд-эффекты или другие неочевидные последствия у этих изменений? Если да, это нужно объяснить здесь.

Параграфы разделяются пустыми строками.

- Можно делать маркированные списки
- Обычно в качестве маркера списка используется звездочка или тире с пробелом перед ними; но тут есть разные соглашения

Если у вас есть баг-трекер [или система управления проектами], поместите ссылки на задачи в конце текста таким образом:

Решено: #123

Смотрите также: #456, #789

► [Оригинал](#)

1. Отделяйте заголовок от тела пустой строкой

Из [мануала](#) к команде **git commit**:

Хотя это не обязательно, но хорошей идеей будет начинать комментарий к коммиту с одной короткой (менее 50 символов) строки, обобщающей сделанные изменения, затем пустая строка и затем более подробное описание. Текст до первой пустой строки в комментарии считается заголовком коммита и используется в разных командах Git. Например, [Git-format-patch\(1\)](#) превращает коммит в email; команда использует заголовок коммита для темы письма и остальной текст — для тела письма.

Во-первых, не каждый коммит требует заполнения и заголовка, и тела. Иногда одной строчки вполне достаточно, особенно когда изменения настолько малы, что никакой дополнительной информации о них не требуется. Например:

```
Fix typo in introduction to user guide
```

Сказанного достаточно; если пользователь захочет узнать, что именно за опечатка была исправлена, он может просто посмотреть на сами изменения, используя **git show** или **git diff**, или **git log -p**.

Если вы коммитите что-то в этом роде с помощью командной строки, удобно будет использовать опцию **-m** для **git commit**:

```
$ git commit -m "Fix typo in introduction to user guide"
```

Однако, когда коммит заслуживает некоторых объяснений и описания ситуации, вам нужно записать их в тело комментария. Например:

```
Derezz the master control program

MCP turned out to be evil and had become intent on world domination.
This commit throws Tron's disc into MCP (causing its deresolution)
and turns it back into a chess game.
```

Комментарии, у которых есть тело, не так удобно писать с помощью опции **-m**. Будет лучше использовать для этого текстовый редактор. Если вы еще не настроили редактор для использования с Git, прочтите [этот раздел книги Pro Git](#).

В любом случае, разделение заголовка и тела комментария окупится при просмотре лога. Вот полная запись о коммите:

```
$ git log
commit 42e769bdf4894310333942ffc5a15151222a87be
Author: Kevin Flynn <kevin@flynnsarcade.com>
Date:   Fri Jan 01 00:00:00 1982 -0200

    Derezz the master control program

    MCP turned out to be evil and had become intent on world domination.
    This commit throws Tron's disc into MCP (causing its deresolution)
    and turns it back into a chess game.
```

А вот команда **git log --oneline**, которая выводит только строку заголовка:

```
$ git log --oneline
42e769 Derezz the master control program
```

Или **git shortlog**, которая группирует коммиты по автору, опять же, для краткости показывает только заголовок:

```
$ git shortlog
Kevin Flynn (1):
    Derezz the master control program

Alan Bradley (1):
    Introduce security program "Tron"

Ed Dillinger (3):
    Rename chess program to "MCP"
    Modify chess program
    Upgrade chess program

Walter Gibbs (1):
    Introduce protoype chess program
```

Есть множество других ситуаций, где необходимо различать заголовок и тело коммита — и для этого они обязательно должны быть разделены пустой строкой.

2. Ограничивайте заголовок 50 символами

Технически, выйти за 50 символов можно, но не рекомендуется. Эта длина заголовка гарантирует его читабельность, а также заставляет автора задуматься о самой краткой и четкой формулировке для описания происходящего.

Подсказка: если вам тяжело резюмировать итоги работы, возможно, в одном коммите содержится слишком много изменений. Стремитесь делать [атомарные коммиты](#) (это тема для отдельного поста).

Интерфейс GitHub'a полностью поддерживает эти соглашения. Он предупредит вас, если вы выйдете за пределы лимита в 50 символов:



Commit changes

 **ProTip:** Great commit summaries are 50 characters or less. Place extra information in the extended description.

Demonstrate a subject line that goes on too long and gets truncated by the GitHub UI

Add an optional extended description...

И обрежет все заголовки длиннее 72 символов, подставив троеточие:



Commits on Aug 31, 2014



Demonstrate a subject line that goes on too long and gets truncated b... ...

cbeams authored 2 minutes ago

Так что стремитесь к 50 символам, но учтите, что 72 — это строгое ограничение.

3. Пишите заголовок с заглавной буквы

Тут все просто. Начинайте все заголовки с заглавной буквы.

Например:

- **Accelerate to 88 miles per hour**

Вместо:

- **accelerate to 88 miles per hour**

4. Не ставьте точку в конце заголовка

В ней нет никакой необходимости. Кроме того, каждый символ на счету, когда мы пытаемся уложиться в 50.

Например:

- **Open the pod bay doors**

Вместо:

- **Open the pod bay doors.**

5. Используйте повелительное наклонение в заголовке

Повелительное наклонение буквально означает: форма глагола, выражающая волеизъявления (приказ, просьбу или совет). Несколько примеров:

- Clean your room (приберись в комнате)
- Close the door (закрой дверь)
- Take out the trash (вынеси мусор)

Каждое из семи правил, о которых вы сейчас читаете, написано в повелительном наклонении («Переходите на следующую строку в теле на 72 символах» и т.д.).

Эта форма может звучать немного грубо, и поэтому не так часто используется [в английском языке — прим. пер.]. Но она идеально подходит для заголовка коммита. Одна из причин — тот факт, что сам Git использует повелительное наклонение, когда создает коммиты от вашего имени.

Например, при использовании **git merge** по умолчанию добавится такое сообщение:

```
Merge branch 'myfeature'
```

А при использовании **git revert**:

```
Revert "Add the thing with the stuff"
```

```
This reverts commit cc87791524aedd593cff5a74532befe7ab69ce9d.
```

Или при клике на кнопку «Merge» в интерфейсе пулл-реквеста в GitHub:

```
Merge pull request #123 from someuser/somebranch
```

Так что, когда вы пишете собственные сообщения к коммиту в повелительном наклонении, вы следуете правилам, заложенным в сам Git. Например:

- **Refactor subsystem X for readability**
- **Update getting started documentation**
- **Remove deprecated methods**
- **Release version 1.0.0**

Такой способ может сначала показаться неудобным. Мы больше привыкли использовать изъявительное наклонение, которое скорее сообщает о фактах. Поэтому сообщения коммитов часто оказываются примерно такими:

- **Fixed bug with Y**
- **Changing behavior of X**

А иногда заголовки просто описывают содержимое коммита:

- **More fixes for broken stuff**
- **Sweet new API methods**

Есть простое правило, которое позволит вам избегать ошибок.

Правильно составленный заголовок коммита должен завершать следующее предложение:

- If applied, this commit will <заголовок коммита>

Например:

- If applied, this commit will **refactor subsystem X for readability**
- If applied, this commit will **update getting started documentation**
- If applied, this commit will **remove deprecated methods**
- If applied, this commit will **release version 1.0.0**

If applied, this commit will **merge pull request #123 from user/branch**

Убедитесь, что глаголы в других, не повелительных, наклонениях здесь не сработают:

- If applied, this commit will **fixed bug with Y**
- If applied, this commit will **changing behavior of X**
- If applied, this commit will **more fixes for broken stuff**

- If applied, this commit will **sweet new API methods**

Помните: использование повелительного наклонения важно только в заголовке коммита. В теле коммита это необязательно.

6. Переходите на следующую строку в теле на 72 символах

Сам Git не расставляет переносы строк автоматически. Редактируя тело комментария, вы должны помнить о правой границе и ставить переносы строк вручную.

Рекомендуется переходить на следующую строку на 72 символах, чтобы Git мог свободно расставлять отступы и всё еще влезать в сумме в 80 символов.

С этим может помочь хороший текстовый редактор. Довольно легко настроить, скажем, Vim, на перенос строки на 72 символах для написания сообщения к коммиту. Однако, так сложилось, что IDE ужасно плохо поддерживают умные переносы строк для сообщений к коммитам (хотя последние версии IntelliJ IDEA [наконец стали лучше](#) в этой части). (*прим. пер. — возможно, на текущий момент все обстоит гораздо лучше*).

7. В теле отвечайте на вопросы «что» и «почему», а не «как»

В этом [коммите из репозитория Bitcoin](#) дано отличное объяснение, что и почему изменилось:

```
commit eb0b56b19017ab5c16c745e6da39c53126924ed6
Author: Pieter Wuille <pieter.wuille@gmail.com>
Date:   Fri Aug 1 22:57:55 2014 +0200

    Simplify serialize.h's exception handling

    Remove the 'state' and 'exceptmask' from serialize.h's stream
    implementations, as well as related methods.

    As exceptmask always included 'failbit', and setstate was always
    called with bits = failbit, all it did was immediately raise an
    exception. Get rid of those variables, and replace the setstate
    with direct exception throwing (which also removes some dead
    code).

    As a result, good() is never reached after a failure (there are
    only 2 calls, one of which is in tests), and can just be replaced
    by !eof().

    fail(), clear(n) and exceptions() are just never called. Delete
    them.
```

Посмотрите на [изменения в коде](#) и подумайте, сколько времени автор сэкономил настоящим и будущим участникам проекта, описав в комментарии контекст проделанной работы. Иначе он, вероятно, был бы утерян навсегда.

В большинстве случаев вы можете опустить подробности того, как именно были внесены изменения. Обычно, код говорит сам за себя в этом смысле (а если он настолько сложен, что требуются пояснения, то для этого есть комментарии в нем самом).

Сосредоточьтесь в первую очередь на пояснении причин, по которым были внесены изменения — опишите ситуацию до внесения изменений (и что было с ней не так), ситуацию после и то, почему вы выбрали именно такой способ решения задачи.

Может быть, в будущем вы поблагодарите сами себя за это!

Подсказки

Полюбите командную строку. Забудьте про IDE.

Есть множество причин — по количеству команд Git — использовать командную строку по максимуму. Git — безумно мощный инструмент;

IDE — тоже, но каждая по-своему. Я ежедневно пользуюсь IntelliJ IDEA, часто имел дело и с другими (например, Eclipse), но никогда не видел чтобы интеграция Git в IDE могла сравниться по простоте и возможностям с командной строкой (как только вы с ней разберётесь).

Определенные возможности IDE по контролю версий просто неоценимы, например автоматическое выполнение `git rm`, когда вы удаляете файл, или других нужных `git`-штуков, когда вы его переименовываете. Но все гораздо хуже, когда вы пытаетесь с помощью IDE сделать коммит слияние, перебазирование (*rebase*) или сложный анализ истории коммитов.

Когда требуется раскрыть полный потенциал Git, командной строке нет равных. Помните, что используете ли вы Bash, Zsh или Powershell — существуют [скрипты автодополнения](#) команд, избавляющие вас от болезненной необходимости запоминать все подкоманды и опции.

Прочтите книгу Pro Git

Великолепная книга [Pro Git](#) (также [на русском языке](#) — прим. пер) есть в свободном доступе. Воспользуйтесь этим!

Метки: [git](#), [cvs](#), [version control](#), [commit message](#), [commit subject](#)

↑

+47

↓

📖


337

👁

24k

💬

114



↑

16,0

↓

Карма

📊

37,6

Рейтинг

👤

6

Подписчики

✉

Написать

📝

Подписаться

@TrebleSnake

Software Developer

Поделиться публикацией

f

🐦

vk

📧

❤

ПОХОЖИЕ ПУБЛИКАЦИИ

8 июля 2011 в 16:54

Ещё раз о «Mercurial против Git» (с картинками)

↑

+71

👁

52,8k

📖

169

💬

129

17 мая 2011 в 15:17

Dropbox как Git репозиторий

↑

+6

👁

16,2k

📖

89

💬

34

1 октября 2009 в 01:50

Линус Торвалдс о GIT на Google Talks [видеоперевод 8 частей]

↑

+97

👁

3,1k

📖

81

💬

42

ЗАКАЗЫ	Фрилансим
Доработать десктоп приложение на JAVA 0 откликов • 0 просмотров	4 000 ₺ за проект
Сделать серверную часть с нуля с api 2 отклика • 29 просмотров	20 000 ₺ за проект
Разработать серверную часть для мобильного приложения 0 откликов • 23 просмотра	25 000 ₺ за проект
Доработка лого для приложения Android 2 отклика • 10 просмотров	500 ₺ за проект
Настройка синхронизации 1С и сайта (Битрикс) 2 отклика • 13 просмотров	3 000 ₺ за проект