Как конфигурировать Git

Какие настройки `git config` сейчас следует устанавливать по умолчанию? Ниже рассмотрены избранные настройки, менять которые не стесняются даже разработчики самого Git.

Несколько недель назад я написал о настройке Git help.autocorrect и поведал странную историю о том, как её значение стали задавать в децисекундах.

Эта статья заставила меня поразмыслить и о других настройках git config, вероятно, не известных широкому кругу пользователей. Возможно, для этих настроек стоит задать по умолчанию иные значения, чем действуют сейчас.

В этом посте я разберу некоторые (пожалуй, малопонятные) настройки Git, которые сам активировал во всех моих проектах. Я подробно расскажу о них, поясню, как они действуют, и почему их, пожалуй, стоит выставить по умолчанию.

Также оказалось, что большинство из изложенных здесь знаний я почерпнул из общения с людьми, чей повседневный труд заключается в поддержке основной базы кода Git.

TLDR

Начнём с вопроса, который, возможно, не особо занимал кого-то из вас, а именно с чудесной и печальной истории значений rerere или ей подобных. Возможно, кого-то из вас посещает мысль «просто сообщите мне настройки — и я не глядя кину их в мой файл `~/.gitconfig`».

Что ж, ценю вашу честность. А теперь начинается интересное:

```
# Явно пойдёт на пользу git


[column]
        ui = auto
[branch]
        sort = -committerdate
[tag]
        sort = version:refname
[init]
        defaultBranch = main
[diff]
        algorithm = histogram
```

```
        colorMoved = plain
        mnemonicPrefix = true
        renames = true
[push]
        default = simple
        autoSetupRemote = true
        followTags = true
[fetch]
        prune = true
        pruneTags = true
        all = true

# чёрт возьми, почему бы и нет?

[help]
        autocorrect = prompt
[commit]
        verbose = true
[rerere]
        enabled = true
        autoupdate = true
[core]
        excludesfile = ~/.gitignore
[rebase]
        autoSquash = true
        autoStash = true
        updateRefs = true

# дело вкуса (раскомментируйте, если не боитесь)

[core]
        # fsmonitor = true
        # untrackedCache = true
[merge]
        # (just 'diff3' if git version < 2.3)
        # conflictstyle = zdiff3
[pull]
        # rebase = true
```

Копипаста, друзья мои.

Как разработчики ядра Git конфигурируют свои Git-ы?

Прежде, чем углубиться в каждую из этих опций по очереди, зададимся интересным вопросом: есть ли среди разработчиков Git наши единомышленники, также считающие, что эти заданные по умолчанию значения стоит поменять?

Этот вопрос не так давно всплыл в почтовой рассылке Git. Честно говоря, о некоторых из этих настроек я сам узнал только из mailing этого треда, озаглавленного «Spring Cleaning» (Весенняя уборка), в котором Фелипе Контрерас бросил вызов команде разработчиков ядра Git. Он спросил, готовы ли те удалить все накопившиеся у них конфигурационные опции и псевдонимы и на своей шкуре попробовать, каково работать с немодифицированным Git, «из коробки».

Он предложил всем участникам рассылки внимательно взвесить, какие настройки они действительно хотят изменить, а также поделиться с сообществом своими топ-листами тех настроек, которые кажутся им наиболее важными.

Результаты оказались очень интересными: получился весьма компактный список из 9 конфигурационных настроек и 3 псевдонимов, по поводу которых все участники эксперимента более-менее сошлись как по новым желательным умолчаниям. Рассмотрим изменения, которые было предложено внести в настройки конфигурации.

```
merge.conflictstyle = zdiff3
rebase.autosquash = true
rebase.autostash = true
commit.verbose = true
diff.colorMoved = true
diff.algorithm = histogram
grep.patternType = perl
feature.experimental = true
branch.sort = committerdate
```

Со времени эксперимента прошло уже 3-4 года, но ни одна из этих настроек не пополнила список умолчаний с тех пор. Здесь интересно, что многие разработчики Git сами изрядно мучаются, когда пытаются работать с Git, не включив хотя бы несколько из этих опций.

Ещё интереснее, что большинство из вас, пожалуй, не знает, что делает любая из этих настроек.

Итак, давайте в них углубимся. Что они делают, и почему вам стоило бы почти в любой ситуации слепо довериться мне и просто включить их?

Сгруппирую эти настройки в три категории:

- Определённо пойдёт на пользу Git
- Чёрт возьми, почему бы и нет?
- Дело вкуса

Что ж, приступим.

Определённо пойдёт на пользу Git

Настройки из первой группы явственно улучшают Git, если включены по умолчанию. Активировав любую из них, вы просто ничего не потеряете.

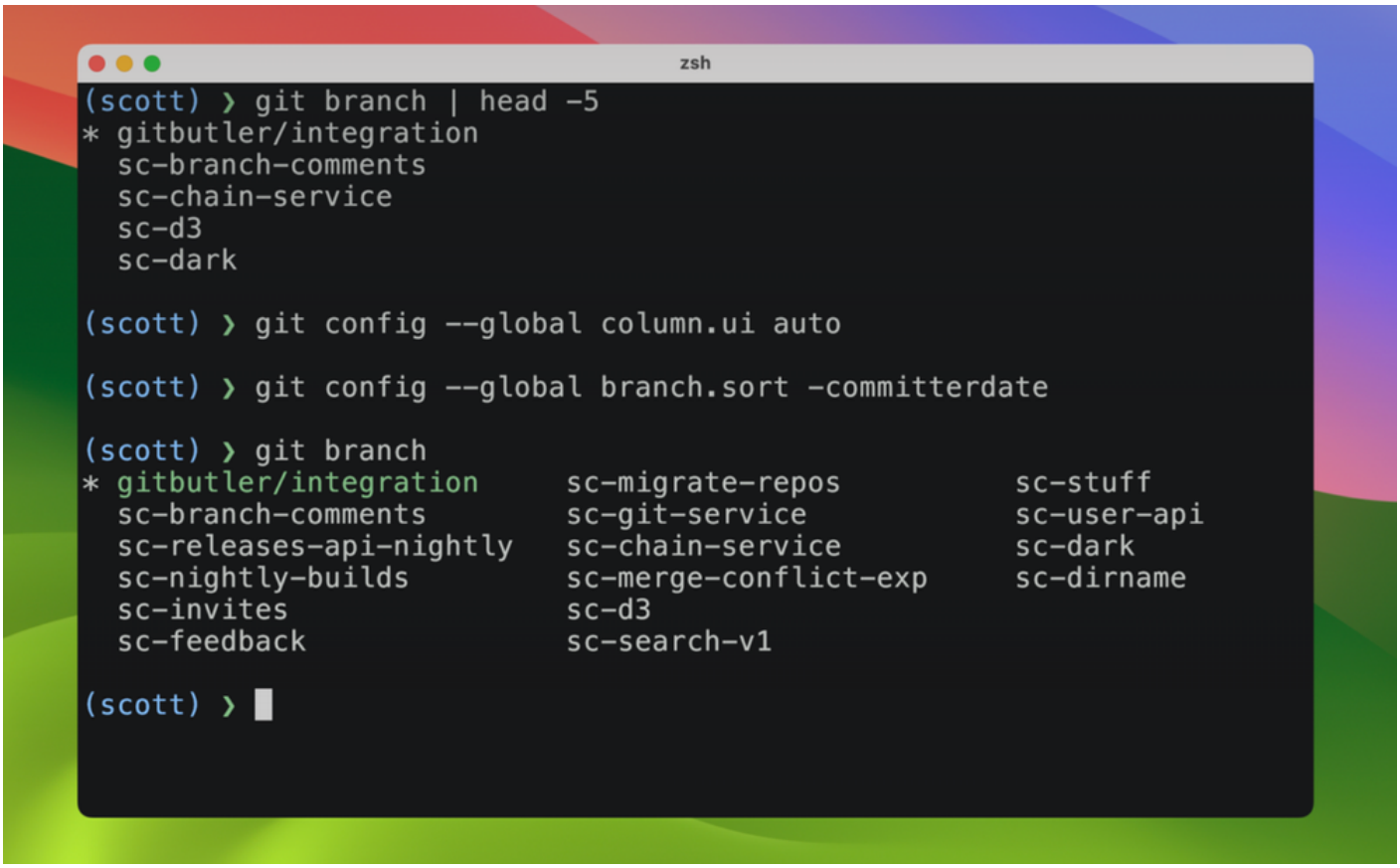
Перечисление веток

В более раннем посте я уже упоминал о «советах Git» (Git Tips) в разделе «Branch Stuff», но, поскольку эта информация была и в списке «Spring Cleaning», думаю, все согласятся: перечисление веток Git по умолчанию не должно упорядочиваться по алфавиту.

Есть две настройки, помогающие улучшить эту ситуацию: `branch.sort` и `column.ui`. Первая из них сортирует список не по алфавиту, а по свежести дат коммитов (поэтому самые интересные коммиты должны находиться в верхней части списка). Вторая выстраивает имена веток в формате столбцов так, чтобы на экране было видно больше такой информации.

```
git config --global column.ui auto
git config --global branch.sort -committerdate
```

Настройка `column.ui` также влияет на вывод других списковых команд (очистка, статус, тег), но в целом я полагаю, что она лучше, чем действующая по умолчанию сейчас.



Можно сортировать записи не только по дате коммита, но я считаю, что это определённо полезнейшая из опций.

Перечисление тегов

Затрагивая тему перечисления, остаётся только удивляться, что опция перечисления тегов по умолчанию не включена — хотя именно она нужна практически кому угодно.

Как правило, попытавшись перечислить теги в алфавитном порядке, вы получите нечто подобное:

```
$ git tag
nightly/0.5.100
nightly/0.5.1000
nightly/0.5.1001
nightly/0.5.101
nightly/0.5.1010
```

Никому не нравится, когда 0.5.101 идёт после 0.5.1000, но таков алфавитный порядок. Можно исправить ситуацию, установив следующую опцию:

```
git config --global tag.sort version:refname
```

Именно этого мы обычно и ожидаем, трактуя версии с указанием номеров через точку как серии целочисленных значений, удобных для сортировки. Поверьте, просто нужно взять и активировать это.

Ветка по умолчанию

Этот пример, возможно, чуть более противоречив, поскольку во многом обусловлен сложившимися привычками. Но в Git лучше предусмотреть для ветки определённое имя по умолчанию, на которое система не будет ругаться всякий раз, когда вы создаёте новый репозиторий при помощи `init`.

```
git config --global init.defaultBranch main
```

Лично меня вполне устраивает master, и я использую такое имя в большинстве из моих репозиториев по умолчанию. Но меня вполне устраивает и main. Поэтому простой выберите наиболее устраивающий вас вариант и установите его.

Мне кажется, просто тупо, что в настоящее время Git отвлекает наше внимание на это, тогда как можно было бы просто обновить значение по умолчанию. Хотелось бы, чтобы в Git в данном случае было лучше со вкусом, поэтому только и остаётся установить тот вариант, который вам самим кажется наиболее разумным. Но выбор за вами.

Улучшенная разность

На самом деле, можно написать целый пост об алгоритмах `git diff`, но, если коротко — по умолчанию Git использует для вычисления разности коммитов старый, быстрый и довольно надёжный «алгоритм Майерса».

Чтобы вы понимали, что такое «старый» — уточню, что этот алгоритм был впервые описан в статье, опубликованной в 1986 году, почти 40 лет назад. Если мы с вами примерно ровесники, могу напомнить, как ощущались те времена. По телевизору тогда показывали комедию «Три амиго», мультфильм «Американский хвост», а в кинотеатрах вышел первый «Горец».

В любом случае, с тех пор были достигнуты некоторые результаты (не без уступок), и вы, возможно, удивитесь, что в базовой комплектации Git встроено 4 готовых к использованию алгоритма вычисления разности: myers, minimal, patience и histogram.

Почти не сомневаюсь, что вы будете пользоваться алгоритмом histogram (инкрементная усовершенствованная версия «patience»), а не «myers», действующим по умолчанию. Можно глобально изменить этот алгоритм вот так:

```
git config --global diff.algorithm histogram
```

Вот пример простого перемещения кода с разницей в myers и histogram, позволяющий составить краткое впечатление о том, насколько умнее можно всё это организовать:

Допустим, мы сдвигаем класс css под другой подобный ему класс, немного его изменяем, а затем выполняем команду `git diff` при помощи алгоритма `myers`, заданного по умолчанию. Получим нечто подобное:

```
❯ git diff
diff --git a/apps/web/src/routes/projects/[projectId]/+page.svelte b
/apps/web/src/routes/projects/[projectId]/+page.svelte
index 7568be2ef..b9e9a00d7 100644
--- a/apps/web/src/routes/projects/[projectId]/+page.svelte
+++ b/apps/web/src/routes/projects/[projectId]/+page.svelte
@@ -141,11 +141,12 @@
 {/if}

 <style>
-       h2 {
+       .event {
                margin-bottom: 1rem;
        }
-       .event {
+       h2 {
                margin-bottom: 1rem;
+               padding: 1px;
        }
        hr {
                margin: 1rem 0;
```

Да, немного запутанно. Вот что в таком же сценарии нам дал бы histogram:

```
❯ git diff
diff --git a/apps/web/src/routes/projects/[projectId]/+page.svelte b
/apps/web/src/routes/projects/[projectId]/+page.svelte
index 7568be2ef..b9e9a00d7 100644
--- a/apps/web/src/routes/projects/[projectId]/+page.svelte
+++ b/apps/web/src/routes/projects/[projectId]/+page.svelte
@@ -141,12 +141,13 @@
 {/if}

 <style>
-       h2 {
-               margin-bottom: 1rem;
-       }
        .event {
                margin-bottom: 1rem;
        }
+       h2 {
+               margin-bottom: 1rem;
+               padding: 1px;
+       }
        hr {
                margin: 1rem 0;
        }
```

Теперь становится немного яснее, что именно здесь произошло.

Не далее, чем в прошлом году Элия (из нашей команды Git Merge) предположил, что histogram или patience лучше подошли бы в качестве вариантов по умолчанию. Фелипе, написавший «Spring Cleaning», в целом предлагал то же самое, но в реальности маловероятно в какой-либо обозримой перспективе этот вызов будет принят.

Здесь много материала, но можно внести в `git diff` и некоторые более мелкие корректировки:

```
git config --global diff.colorMoved plain
git config --global diff.mnemonicPrefix true
git config --global diff.renames true
```

Вариант colorMoved также был в списке предложений из «Spring Cleaning», поэтому данное изменение также, пожалуй, следует внести по умолчанию.

Рассмотрим предыдущий пример с переносом кода при включённой опции `colorMoved`:

Вы видите, чем отличается перемещённый код и добавленная строка. При включённой опции colorMoved перемещение кода будет отображаться разными цветами, в зависимости от того, были какие-то строки добавлены или удалены.

Опция diff.renames позволяет выявить, был ли файл переименован, и обычно это хорошо (пусть и немного затратно), а diff.mnemonicPrefix заменяет `a/` и `b/` в выведенном заголовке разности фрагментов на информацию о том, откуда пришло различие, вот так: `i/` (индекс), `w/` (рабочий каталог) или `c/` (коммит).

Таким образом, если я сравниваю изменение из моего индекса с информацией из моего рабочего каталога, то получу в качестве заголовка изменённого фрагмента следующий вывод:

```
❯ git diff
diff --git i/apps/web/page.js w/apps/web/page.js
index 7568be2ef..b9e9a00d7 100644
--- i/apps/web/page.js
+++ w/apps/web/page.js
```

Может быть, из этого примера не вполне очевидно, но по именам путей вполне можно уловить, какая часть поступила из индекса, а какая — из рабочего каталога. Разница в самом деле тонкая, но тем она мне и нравится.

Улучшаем пуш

Среди тех вещей, которые неизменно путают и напрягают меня с первых дней работы с Git — это сложности с правильной настройкой отслеживания веток. Когда я пушу код, куда пушится этот код, и пушится ли вообще?

Есть три обновлённые настройки, задавая которые по умолчанию, можно значительно повысить удобство работы с инструментом. Первая (`push.default simple`) назначена по умолчанию, начиная с Git 2.0, но другие по-прежнему нужно устанавливать явно.

```
git config --global push.default simple # (по умолчанию с версии 2.0)
git config --global push.autoSetupRemote true
git config --global push.followTags true
```

Этот момент всегда был в Git болевой точкой. Новое умолчание simple более или менее рассчитано на работу с централизованными потоками задач, стандартно пуш осуществляется в текущую ветку и в одноимённую ветку на удалённой машине. Я думаю, это очень разумное умолчание.

Однако, если эта ветка не существует, а отслеживание веток не настроено, то вы всё равно получите следующую ошибку:

```
$ git push
fatal: The current branch my-branch-name has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin my-branch-name
```

Вполне возможно, что следующую картинку вы видели миллион раз.

Если установить `push.autoSetupRemote` в `true`, то эту ошибку вы более получать не будете. Если вышестоящая ветка не установлена, то программа её автоматически установит. Просто не передать, как мне нравится эта настройка.

Наконец, настройка `push.followTags` запушит все теги, имеющиеся у вас на локальной машине, но отсутствующие на сервере. Это происходит при каждой пуш-операции. Я на этом несколько раз обжигался. Но, если вы создаёте теги на локальной машине, то активируйте эту настройку, тогда сможете не беспокоиться, что ваших тегов не увидят другие люди.

Улучшенная выборка

Можно утверждать, что было бы хорошо держать на локальной машине историю копий веток и тегов, которые когда-то были на сервере, а теперь там отсутствуют, но я с этим согласиться не могу.

Лично я считаю, что по умолчанию Git должен работать так: держать ссылки на ваш код, расположенные на локальной машине, максимально схожими на ссылки, находящиеся на удалённой машине. Отсекайте всё устаревшее, т.д.

Поэтому я поставил бы по умолчанию следующие настройки выборки:

```
git config --global fetch.prune true
git config --global fetch.pruneTags true
git config --global fetch.all true
```

На самом деле, всё это позволяет нам не сомневаться, что мы удалим `origin/blah`, если `blah` будет удалено на сервере. Причём всё это будет автоматически делаться на всех удалённых машинах, если правильно сконфигурировать такие настройки. Мне кажется, это очень разумно.

Чёрт возьми, почему бы и нет?

Настройки из следующей группы, как правило, безобидны, но могут и помочь.

Я не уверен, что здесь обязательно менять умолчания, но и не думаю, что приведённые здесь варианты кому-то повредят. В некоторых случаях они даже очень пригодятся. Поэтому включу их в мой список.

Предложения автозамены

Как я подробно объяснял в одном из моих предыдущих постов, в Git есть такая довольно приятная возможность: если случайно механически ошибётесь при наборе команды, система попробует догадаться, что вы имели в виду, и запустить именно нужную команду.

По умолчанию мы попытаемся этого избежать, а сделать наоборот: чтобы программа сама предлагала вам, что ввести.

```
git config --global help.autocorrect prompt
```

Если вы хотите подробнее почитать об этой настройке, чем она обоснована, а также какова её история, то вот дотошный пост о ней.

Коммиты с указанием разности

Это предложение содержалось в списке «Spring Cleaning» — думаю, оно попало туда в основном потому, что с его помощью можно расширить контекст и добавить информацию, с которой затем можно будет сверяться, когда будете писать в редакторе сообщения о коммитах.

По умолчанию `git commit` будет выдавать сообщение, которое выглядит примерно так:

Здесь мы видим только список изменившихся файлов. Если установить `commit.verbose` в значение `true`, то сюда будет записан весь вывод `diff`, и вы сможете сверяться с ним, когда будете писать ваше сообщение.

```
git config --global commit.verbose true
```

Вот какой вид это примет теперь, когда вы перейдёте к коммиту:

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Your branch is behind 'origin/master' by 126 commits, and can be fast-forwarded.
#   (use "git pull" to update your local branch)
#
# Changes to be committed:
#       modified:   apps/web/src/routes/projects/[projectId]/+page.svelte
#
# ------------------------ >8 ------------------------
# Do not modify or remove the line above.
# Everything below it will be ignored.
diff --git c/apps/web/src/routes/projects/[projectId]/+page.svelte i/apps/web/src/r
index 7568be2ef..b9e9a00d7 100644
--- c/apps/web/src/routes/projects/[projectId]/+page.svelte
+++ i/apps/web/src/routes/projects/[projectId]/+page.svelte
@@ -141,12 +141,13 @@
 {/if}

 <style>
-       h2 {
-               margin-bottom: 1rem;
-       }
        .event {
                margin-bottom: 1rem;
        }
+       h2 {
+               margin-bottom: 1rem;
+               padding: 1px;
+       }
        hr {
                margin: 1rem 0;
        }
```

Всё это будет удалено из сообщения о коммите (всё, что расположено ниже этой забавной «линии разреза» `-- >8 --`), но зато вы получите гораздо более подробный контекст, когда будете писать сообщение.

Повторное использование ранее записанных разрешений конфликтов

Эта настройка пригодится вам лишь в случаях, когда вы выполняете перебазирование, и при этой работе раз за разом возникают конфликты. Не самая распространённая ситуация, но не будет и никаких проблем, если эту настройку просто включить и никогда не использовать.

```
git config --global rerere.enabled true
git config --global rerere.autoupdate true
```

Опция `enabled` гарантирует, что система будет записывать состояния «до» и «после» конфликтов при перебазировании. После этого `autoupdate` будет автоматически раз за разом

применять проверенные разрешения конфликтов, с которыми сталкивается не в первый раз.

Я достаточно подробно написал об этом здесь, поэтому не буду здесь заново утомлять вас этой темой.

Глобальный файл ignore

Эта настройка даже не простая, а тупая, но, поскольку есть файл `~/.gitconfig` с глобальными значениями, было бы круто иметь и файл `~/.gitignore` с глобальными значениями. Благодаря этой настройке достигается следующее:

```
git config --global core.excludesfile ~/.gitignore
```

На самом деле, в этом, мягко говоря, нет необходимости, поскольку Git и так ищет глобальные ignore-значения в двух следующих местах: `~/git/ignore` и `~/.config/git/ignore`. Но, поскольку они выглядят немного непонятно, полагаю, было бы хорошо проложить туда более логичный путь.

Чуть более аккуратное перебазирование

В этом разделе рассмотрим преимущественно тот практический случай, в котором вы исправляете и ужимаете ваши комментарии. Если вы не знаете, что это такое, можете почитать мой более ранний пост на тему autosquashing.

Правда, если этой работой приходится заниматься в больших масштабах (или даже время от времени), вам определённо помогут и не повредят следующие настройки.

```
git config --global rebase.autoSquash true
git config --global rebase.autoStash true
git config --global rebase.updateRefs true
```

Настройка `updateRefs` практически наверняка должна выставляться по умолчанию, честно. Она просто организует в виде ветки стопку ссылок и гарантирует, что при перебазировании ветки эти ссылки также будут правильно перемещены.

Дело вкуса

Следующая группа настроек зависит от ваших личных предпочтений. Но большинство не знает, что эти настройки существуют, и многим они могут оказаться полезными. В разделе TLDR выше в этой статье я их закомментировал.

Улучшенная обработка конфликтов при слиянии

Итак, хотя эта тема и затрагивается в треде «Spring Cleaning» как потенциальная новая настройка по умолчанию, я не уверен, что с этим можно согласиться.

Когда в Git возникает конфликт при слиянии веток, лучше не расставлять маркеры конфликтов слева и справа, можно попросить вставить ветку, которая выглядела бы как новая база. Иногда это бывает по-настоящему полезно, но некоторых весьма раздражает.

```
git config --global merge.conflictstyle zdiff3
```

В почтовой рассылке Git шли дискуссии о том, не активировать ли эту настройку по умолчанию. Кстати, в GitButler при работе с маркерами конфликтов используется стратегия `diff3` — и, будем честны, далеко не всем она нравится.

Рассмотрим пример с установкой простого маркера при конфликте слияния, который может возникнуть в файле при слиянии или перебазировании:

```
1  #!/usr/bin/env ruby
2
   Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Char
3  <<<<<<< HEAD (Current Change)
4  print "hello america"
5  =======
6  print "hola mundo"
7  >>>>>>> spanish (Incoming Change)
8
```

Если включить настройку `merge.conflictStyle zdiff3`, получим:

```
1  #!/usr/bin/env ruby
2
   Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Cha
3  <<<<<<< HEAD (Current Change)
4  print "hello america"
5  ||||||| edb789f
6  print "hello world"
7  =======
8  print "hola mundo"
9  >>>>>>> spanish (Incoming Change)
10
```

В сущности, кроме секций <<<<<< и >>>>>>, в которых вы видите, как вы изменили блок, и как его изменил кто-то другой, здесь также добавляется блок |||||||, демонстрирующий, как выглядела ветка до вышеупомянутых изменений.

Этот дополнительный контекст (позволяющий судить, как выглядела ветка до того, как в неё были внесены изменения) иногда бывает крайне полезен, но, как правило, это всего лишь дополнительные данные, в которых можно запутаться.

На самом деле, именно от вас зависит, хотите ли вы иметь под рукой эти дополнительные данные.

Скрытый текст

Стратегия `diff3` применялась в Git почти всегда. Я рекомендую здесь zdiff3, что означает "ревнивое `diff3`" и чуть лучше на практике, но этот вариант доступен только начиная с версии Git 2.35 (январь 2022). Если вы работаете с более старой версией Git — просто уберите "`z`".

Улучшенное подтягивание

Конечно, в споре о слиянии и перебазировании едва ли можно прийти к общему мнению, но, как правило, у нас есть предпочтения. Возможно, вы не знали, что, если установить `git pull` по умолчанию, то делаться будет только один или другой вариант. Нет необходимости в `git pull --rebase`, можно просто задать по умолчанию следующее:

```
git config --global pull.rebase true
```

Это только вам решать, но я лишь недавно с концами перебрался в лагерь «только перебазирование», конечно, это отразилось на той конфигурации, которой я пользуюсь.

Выполняйте процессы fsmonitor

Опять же, это актуально только для больших репозиториев, возможно, вы не хотите, чтобы в вашей файловой системе повсюду шёл мониторинг. Но эти процессы могут значительно ускорить такие вещи как git status, если вам приходится иметь дело с большими рабочими каталогами.

Может быть, эту настройку и не стоит задавать по умолчанию, но она неплоха, и в определённых случаях, что называется «делает разницу». Может быть, стоит спрашивать в `git clone`, хотите ли вы её задействовать. В любом случае, такая опция существует.

```
git config --global core.fsmonitor true
git config --global core.untrackedCache true
```

Так будет запускаться процесс для мониторинга файловой системы (по одному на каждый репозиторий), который будет замечать изменения в файлах и обновлять кэш, так, что `git status` не придётся заново индексировать каждый файл тысячами статистических вызовов `mtime`, лишь чтобы проверить, не появились ли элементарные изменения в логах.

Подчеркну, что в данном случае запускается по одному процессу на каждый репозиторий, в котором вы работаете — может получиться немало. Обычно, правда, работают они не так активно, поскольку основаны на событиях, и поэтому они не оказывают заметной нагрузки на память или ЦП, даже если таких процессов сотни. Но стоит иметь в виду такую возможность. Также просто можете убрать опцию `--global` и активировать её только в больших репозиториях.

Заключение

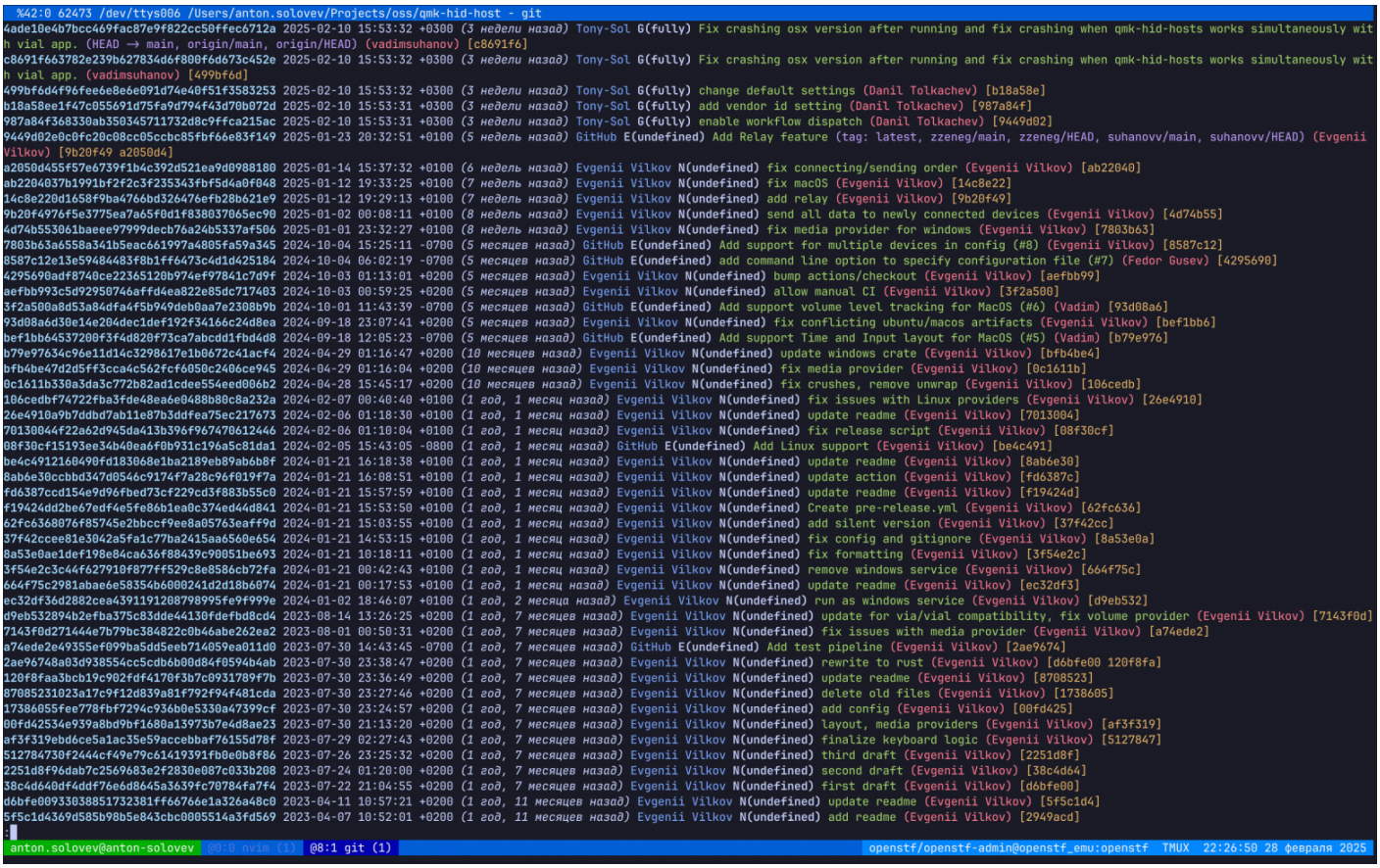
Надеюсь, эта статья послужит вам полезной справкой. Возможно, вы узнали что-то новое о конфигурации в Git и согласитесь со мной, что хотя бы некоторые из перечисленных вещей обязательно нужно выставлять по умолчанию. По некоторым из этих вариантов даже в почтовой рассылке сообщества Git царит единогласие.

Есть и множество других способов пришпорить Git (псевдонимы, классные внешние инструменты pager и diff, другие подобные вещи), но я думаю, что вам было бы лучше всего придерживаться глобально полезных и относительно простых стандартных настроек Git.

P.S. В конфигурационный файл для соблюдения правила "1 строка = 1 запись" можно добавить:

```
[format]
	pretty = format:%C(cyan bold)%H%C(reset) %ci %C(italic)(%cr)%C(reset) %C(blue)%cn%C(reset)
%C(auto,bold)%G?(%GT)%C(reset) %C(green)%s%C(magenta)%d%C(reset) %C(red)(%an)%C(reset)
%C(yellow)[%p]%C(reset)
```

Выглядит так:



Revision #3
Created 2025-11-17 14:11:24 UTC by Эд Кукса
Updated 2025-11-27 18:35:51 UTC by Эд Кукса