6 декабря 2012 в 13:31

Git Rebase: руководство по использованию

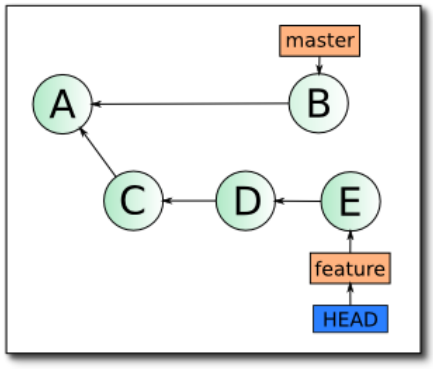
tutorial

Git*

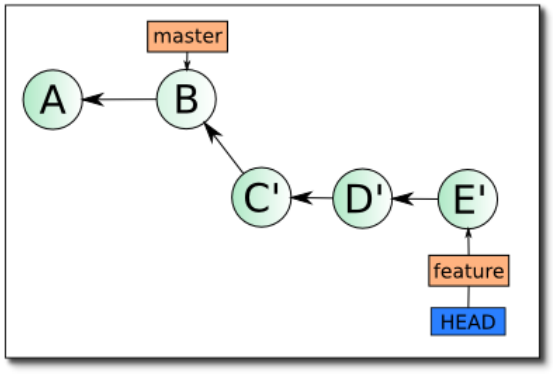
Rebase — один из двух способов объединить изменения, сделанные в одной ветке, с другой веткой. Начинающие и даже опытные пользователи git иногда испытывают нежелание пользоваться ей, так как не видят смысла осваивать еще один способ объединять изменения, когда уже и так прекрасно владеют операцией merge. В этой статье я бы хотел подробно разобрать теорию и практику использования rebase.

Теория

Итак, освежим теоретические знания о том, что же такое rebase. Для начала вкратце — у вас есть две ветки — **master** и **feature**, обе локальные, feature была создана от master в состоянии A и содержит в себе коммиты C, D и E. В ветку master после отделения от нее ветки feature был сделан 1 коммит B.



После применения операции rebase master в ветке feature, дерево коммитов будет иметь вид:



Обратите внимание, что коммиты C', D' и E' — не равны C, D и E, они имеют другие хеши, но изменения (дельты), которые они в себе несут, в идеале точно такие же. Отличие в коммитах обусловлено тем, что они имеют другую базу (в первом случае — A, во втором — B), отличия в дельтах, если они есть, обусловлены разрешением конфликтных ситуаций, возникших при rebase. Об этом чуть подробнее далее.

Такое состояние имеет одно важное преимущество перед первым, при слиянии ветки feature в master ветка может быть объединена по fast-forward, что исключает возникновение конфликтов при выполнении этой операции, кроме того, код в ветке feature более актуален, так как учитывает изменения сделанные в ветке master в коммите B.

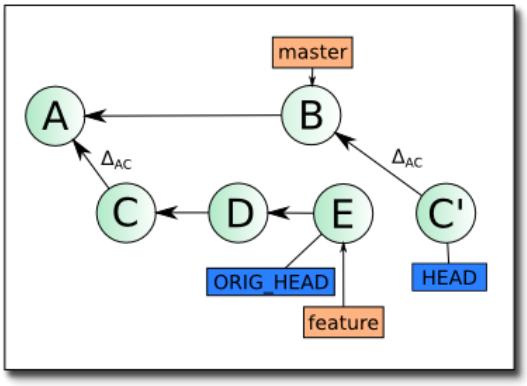
Процесс rebase-а детально

Давайте теперь разберемся с механикой этого процесса, как именно дерево 1 превратилось в дерево 2?

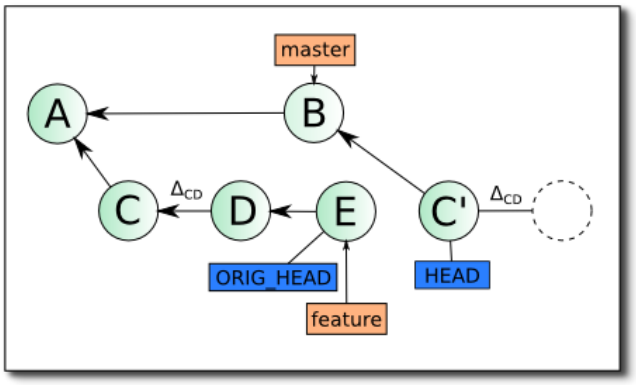
Напомню, перед rebase вы находтесь в ветке feature, то есть ваш HEAD смотрит на указатель feature, который в свою очередь смотрит на коммит E. Идентификатор ветки master вы передаете в команду как аргумент:

```
git rebase master
```

Для начала git находит базовый коммит — общий родитель этих двух состояний. В данном случае это коммит A. Далее двигаясь в направлении вашего текущего HEAD git вычисляет разницу для каждой пары коммитов, на первом шаге между A и C, назовем ее Δ_{AC}. Эта дельта применяется к текущему состоянию ветки master. Если при этом не возникает конфликтное состояние, создается коммит C', таким образом C' = B + Δ_{AC}. Ветки master и feature при этом не смещаются, однако, HEAD перемещается на новый коммит (C'), приводя ваш репозитарий состояние «отделеной головы» (detached HEAD).



Успешно создав коммит C', git переходит к переносу следующих изменений — Δ_{CD}. Предположим, что при наложении этих изменний на коммит C' возник конфликт. Процесс rebase останавливается (именно в этот момент, набрав git status вы можете обнаружить, что находитесь в состоянии detached HEAD). Изменения, внесенные Δ_{CD} находятся в вашей рабочей копии и (кроме конфликтных) подготовлены к коммиту (пунктиром обозначена stage-зона):



Далее вы можете предпринять следующие шаги:

1. Отменить процесс rebase набрав в консоли

```
git rebase --abort
```

При этом маркер HEAD, будет перенесен обратно на ветку feature, а уже добавленные коммиты повиснут в воздухе (на них не будет указывать ни один указатель) и будут вскоре удалены.

2. Разрешить конфликт в вашем любимом merge-tool'e, подготовить файлы к коммиту, набрав `git add %filename%`. Проделав это со всеми конфликтными файлами, продолжить процесс rebase-а набрав в консоли

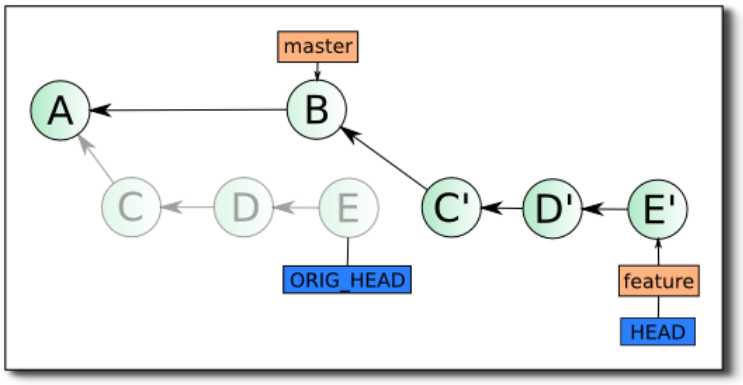
```
git rebase --continue
```

При этом, если все конфликты действительно разрешены, будет создан коммит D' и rebase перейдет к следующему, в данном примере последнему шагу.

3. Если изменения, сделанные при формировании коммита B и коммита D являются полностью взаимоисключающими, причем «правильные» изменения сделаны в коммите B, то вы не сможете продолжить набрав `git rebase --continue`, так как разрешив конфликты обнаружите, что изменений в рабочей копии нет. В данном случае вам надо пропустить создание коммита D', набрав команду

```
git rebase --skip
```

После применения изменений Δ_{DE} будет создан последний коммит E', указатель ветки feature будет установлен на коммит E', а HEAD станет показывать на ветку feature — теперь, вы находитесь в состоянии на втором рисунке, rebase окончен. Старые коммиты C, D и E вам больше не нужны.



При этом коммиты, созданные в процессе rebase-а, будут содержать данные как об оригинальном авторе и дате изменений (Author), так и о пользователе, сделавшем rebase (Commiter):

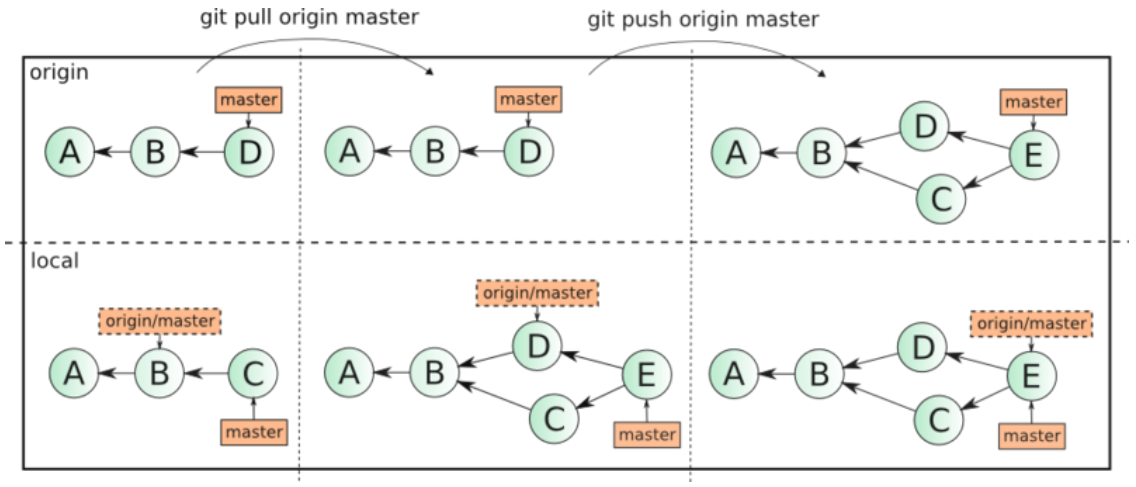
```
commit 0244215614ce6886c9e7d75755601f94b8e19729
Author:     sloot69 <***@****.com>
AuthorDate: Mon Nov 26 13:19:08 2012 +0400
Commit:     Alex <***@****.com>
CommitDate: Mon Nov 26 13:33:27 2012 +0400
```

С небес на землю — rebase в реальных условиях

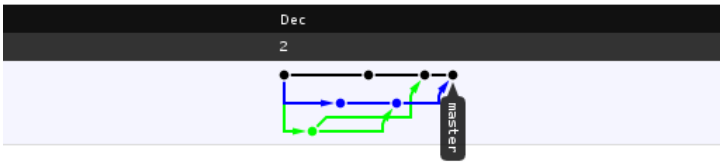
На самом деле обычно вы работаете не с двумя ветками, а с четырьмя в самом простом случае: master, origin/master, feature и origin/feature. При этом rebase возможен как между веткой и ее origin-ом, например feature и origin/feature, так и между локальными ветками feature и master.

Rebase ветки с origin-ом

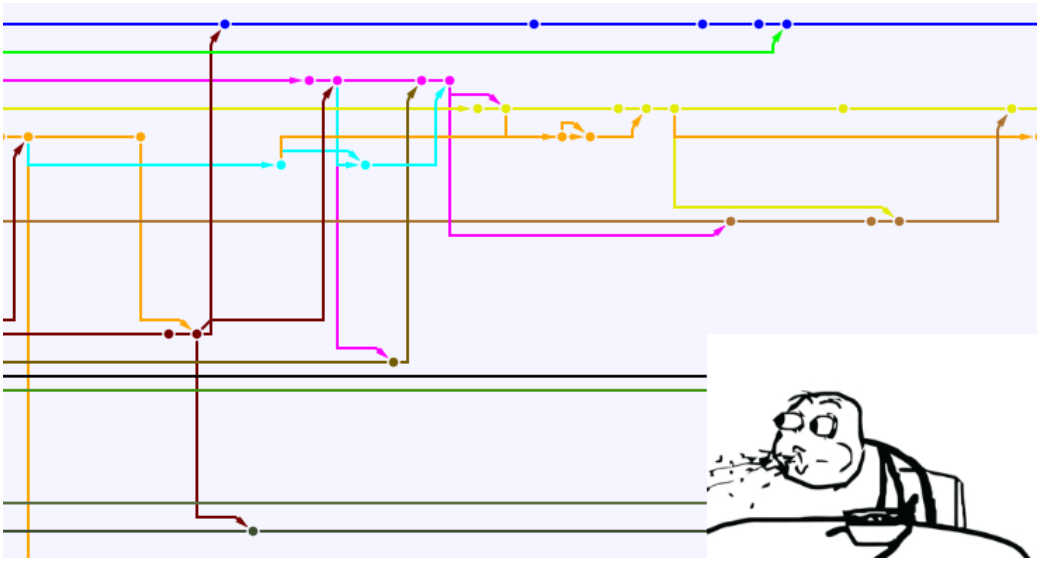
Если вы хотите начать работать с rebase, то лучше всего начать с ребейза своих изменений в ветке относительно ее копии в удаленном репозитарии. Дело в том, что когда вы добавляете коммит, и в удаленном репозитарии добавляется коммит, для объединения изменений по-умолчанию используется merge. Выглядит это примерно так:



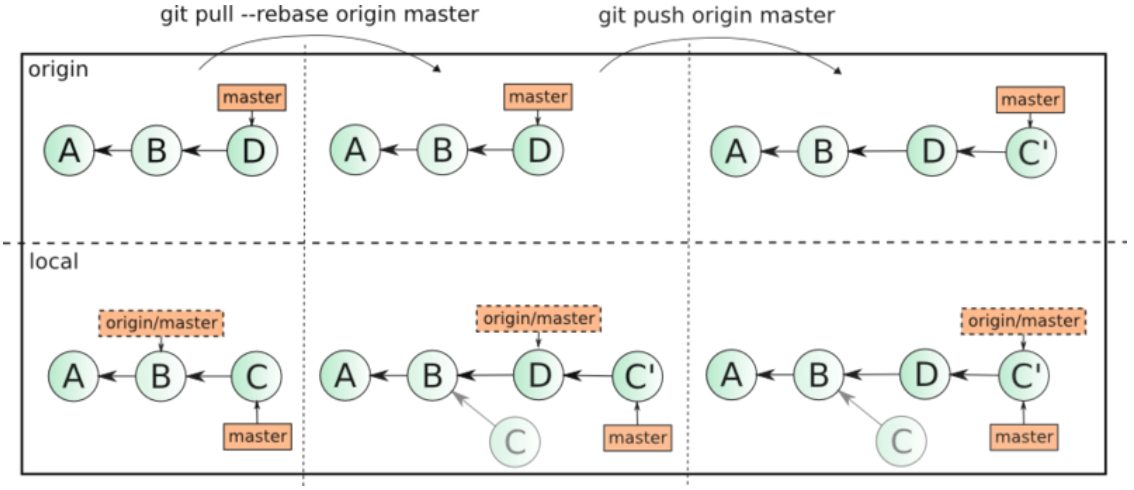
Представим умозрительную ситуацию — 3 разработчика активно работают с веткой master в удаленном репозитарии. Делая одновременно комиты на своих машинах они отправляют каждый по 1 изменению в ветку. При этом первый отправляет их без проблем. Второй и третий сталкивается с тем что ветка не может быть отправлена операцией `git push origin master`, так как в ней уже есть изменения, которые не синхронизированы на локальные машины разработчиков. Оба разработчика (2 и 3) делают `git pull origin master`, создавая при этом локальные merge-коммиты у себя в репозитарии. Второй делает git push первым. Третий при попытке отправить изменения снова сталкивается с обновлением удаленной ветки и снова делает git pull, создавая еще один merge-коммит. И наконец, третий разработчик делает успешный `git push origin master`. Если удаленный репозитарий расположен например на github, то network, то есть граф коммитов будет иметь следующий вид:



Три коммита превратились в 6 (базовый коммит не считаем), история изменений неоправдано запутана, информация об объединении локальных веток с удаленными, на мой взгляд, лишняя. Если масштабировать эту ситуацию на несколько тематических веток и большее количество разработчиков, граф может выглядеть, например, так:



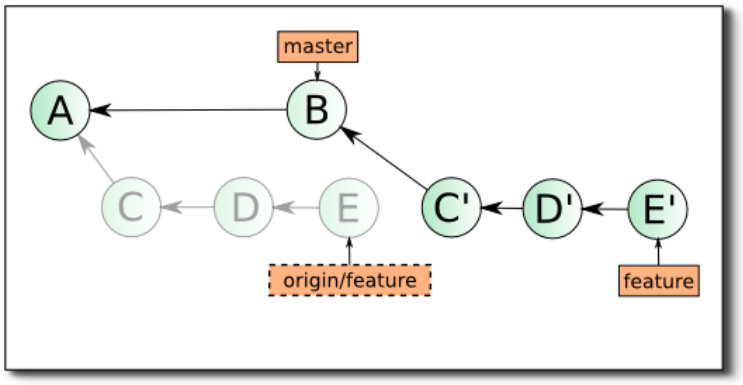
Анализ изменений в таком графе неоправданно трудоемкое занятие. Как тут может помочь rebase?
Если вместо `git pull origin master` выполнить `git pull --rebase origin master`, то ваши локальные изменения, подобно коммитам C, D и E из первого примера будут перенесены наверх обновленного состояния ветки origin/master, позволяя без создания дополнительных коммитов передать их на удаленный сервер с помощью `git push origin master`. То есть слияние изменений будет выглядеть уже так:



Как видно, «лишних» merge-коммитов создано не было, ваши изменения в коммите C были использованы для создания коммита C'. Сам же коммит C остался у вас в локальном репозитарии, в удаленный репозитарий был передан только C'. Если все программисты в команде возьмут за правило пользоваться `git pull --rebase`, тогда каждая из веток в удаленном репозитарии будет выглядеть линейно.

Как поделиться веткой, к которой применен rebase с коллегой

Подробно процесс rebase-а локальной тематической ветки отосительно ветки master был рассмотрен в самом начале статьи. Оговорюсь только, что процесс rebase содержит количество шагов равное количеству коммитов в вашей локальной ветке. Вероятность потенциальных конфликтов, как и в случае с merge, растет пропорционально росту количества коммитов в базовой ветке (master). Поэтому лучше периодически проводить rebase для долгоживущих тематических веток. Но если тематическая ветка имеет свой оригинал на удаленном сервере, как передать ветку в удаленный репозитарий? Если делать push можно только для изменений, которые могут быть приняты по fast-forward, а в данном случае, как мы знаем, это не так:



Тут все просто, наберите в консоли команду:

```
git push origin feature --force
```

Force-режим просто копирует отсутствующие родительские коммиты ветки feature на origin и насильно устанавливает указатель ветки на тот же коммит, что и ваш локальный.

Будьте внимательны! Если вы забудете указать идентификатор ветки, то force-push будет выполнен для всех локальных веток, имеющих удаленный оригинал. При этом нужно понимать, что некоторые локальные ветки могут быть в неактуальном состоянии. То есть **измененения, которые вы не успели затянуть будут удалены в origin-е**. Конечно, сами коммиты не будут удалены — сбросятся только указатели ветки. Эта ситуация поправима — достаточно для каждой ветки найти человека, который последним пушил изменения в нее или уже успел их забрать. Он может сделать обычный push, вновь передав их на origin. Но вся эта морока вам ни к чему, так что лучше просто будьте внимательны.

Ваш коллега, находится в той же ситуации перед pull, в которой вы находились перед тем как сделали push. Только позиции feature и origin/feature отличаются с точностью до наоборот. Если он выполнит обычный `git pull origin feature`, то произойдет попытка объединения старой и новой ветки с помощью merge. Так как старая и новая ветки содержат одни и те же изменения, то все они будут конфликтными. В данном случае нам снова поможет команда:

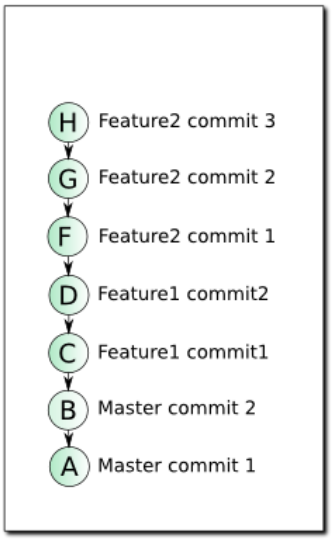
```
git pull --rebase origin feature
```

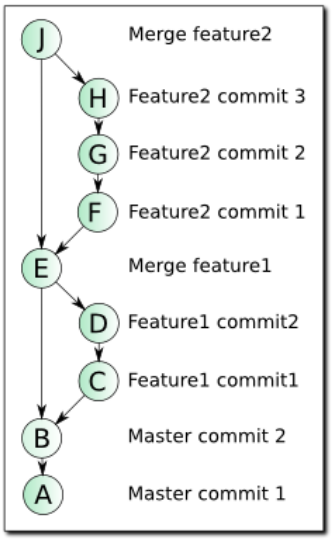
Она заберет новую ветку feature и переместит локальные изменения вашего коллеги на ее вершину.
Вообще `git pull --rebase origin feature` — это безпроигрышный вариант, так как если rebase не требуется, произойдет обычное объединение указателей по fast-forward.

Реинтеграция тематической ветки в master

Мы рассмотрели все необходимые операции для работы с ветками в стиле rebase. Осталось только переключиться в ветку master и сделать git merge feature. Ветка, подготовленная rebase-ом вольется в master по fast-forward, то есть указатель будет просто перемещен вперед.

Однако, у такого подхода есть один недостаток — после merge по fast-forward будет затруднительно определить, какие коммиты были сделаны в вашей ветке, а какие в другой ветке, влитой перед вашей или непосредственно в мастере. Хорошим тоном может быть принятие в вашей команде правила — при интеграции изменений из тематической ветки в основную применять операцию merge с опцией --no-ff. При этом будет создан merge-коммит, одним из родителей которого будет позиция ветки master, другим — позиция ветки feature. Сравните граф коммитов, при объединении веток по fast-forward (слева) и с merge-коммитами, полученными с помощью опции --no-ff (справа):





В данном случае, на мой взгляд, merge-коммиты полезны и несут в себе информацию о моменте объединения веток. Этот граф выглядит как учебный пример, но такая структура вполне реальна при соблюдении некоторых простых правил всеми членами команды.

Заключение

Мы видим, что читаемость графа изменений может быть улучшена на порядок при соблюдении нескольких простых правил, хотя они и требуют небольших дополнительных временных затрат.

В официальном руководстве по git крайне не рекомендуют применять rebase к ветке, которая была запушена в публичный репозитарий, приводятся проблемы, связанные с merge-ем старой ветки, к которой еще не применен rebase (хранящейся у кого-то, кто успел ее загрузить из репозитария) и новой ветки, что приводит к конфликтам и появлению коммитов с одинаковыми датами и сообщениями в логе изменений, здесь мы рассмотрели, как легко можно избежать этой проблемы с помощью git pull --rebase.

В данной статье сделано одно допущение. Все это верно при простой модели ветвления — есть одна главная ветка master и несколько тематических, которые создаются от нее. Когда от тематической ветки создается другая тематическая ветка, есть свои нюансы при rebase-е первичной и вторичной ветки. О них можно прочитать в том самом официальном руководстве.

Иногда споры, что же лучше merge или rebase доходят до холивара. От себя могу сказать, что в конечном счете выбор за вами, однако этот выбор не может быть продиктован уровнем владения тем или иным инструментом. Обоснованный выбор можно сделать, только когда для вас не составит труда работать и в том и в другом стиле. Я не агитирую за повсеместное использование rebase-а, а просто объясняю как им пользоваться. Надеюсь, это статья поможет вам снять вопросы, связанные с механизмом работы rebase и его применением в ежедневной работе.

PS. Хочу поблагодарить своих коллег, продуктивные беседы с которыми позволили мне лучше разобраться в материале, положенном в основу этой статьи.

git, rebase, scm

+120 112434 819 Dr_Logic 24,1

А как ты используешь новые измерения в создании приложений?

Похожие публикации

Полуавтоматическое инкрементирование версии проекта при работе с GIT в Visual Studio 21 сентября в 19:35

Странный глюк Git, чуть не стоивший 10 часов работы 10 сентября в 15:01

Интегрируем Git в Sublime Text 19 августа в 19:58

6 мифов, мешающих разработчикам использовать Git 31 июля в 23:14

Перенос кода с tfs на git 2 июля в 07:36

Git 2.0.0 4 июня в 18:56

Синхронизация структуры MySQL + Git 8 мая в 15:22

Git rebase «по кнопке» 11 сентября 2013 в 16:58

Чем опасен rebase-2, или как rebase мешал баг искать 15 мая 2013 в 16:00

Чем опасен rebase, или как получилось, что 2*3=5 8 мая 2013 в 19:07

Комментарии (169)

k12th, 6 декабря 2012 в 13:50 # +1

У rebase есть одна хорошая вещь — он может squash'ить коммиты, т.е. объединять несколько в один.
Однажды это меня здорово спасло, когда коллега запушил файлы с виндовыми окончаниями строк вместо юниксовых (т.е. весь файл считался измененными и невозможно было понять, что же поменялось).

Dr_Logic, 6 декабря 2012 в 13:53 # ↑ +2

Да, про интерактивный режим ребейза стоит написать отдельную статью)) Тут я рассматривал rebase именно как механизм слияния изменений. Вообще его возможности богаче.

beono, 6 декабря 2012 в 17:07 # ↑ +1

А что скажите об *autocrlf = true*
Просто сам спасаюсь именно этой настройкой, но думаю вдруг чего упустил.

k12th, 6 декабря 2012 в 17:10 # ↑ 0

Прекрасная настройка, но проблема в том, что коммит уже был сделан, когда мы обнаружили, что у коллега она не включена.

fox_anthony, 6 декабря 2012 в 21:45 # ↑ 0

.gitattributes хорошая штука. Можно указать какие переносы строк в каких файлах использовать. Правда это все 100% работает если Вы используете git (а не JGit — насколько я помню там еще эта штука не поддерживается — поправьте, если уже да). Разница между настройкой и .gitattributes в том, что файлик находится в репозитории и там уже тяжело ошибиться. При коммите он насильственно будет использовать нужные переводы строк, что собственно и нужно. Ошибиться довольно тяжело.

k12th, 6 декабря 2012 в 21:48 # ↑ 0

Не слышал, спасибо за инфу.
А что, она помогает пост-фактум, после коммита?

fox_anthony, 6 декабря 2012 в 21:52 # ↑ 0

После коммита нет конечно же =). Но гарантирует, что в будущем таких ситуаций не будет. Использую всегда в своих проектах.

k12th, 6 декабря 2012 в 22:02 # ↑ 0

Хорошая вещь. Но мне нужно было поправить пост-фактум.

Сделал коммит с исправлением окончаний строк, потом засквошил коммиты коллеги и свой в один.

alexanderzaytsev, 7 декабря 2012 в 01:08 # ↑ 0

Можно было сделать через filter-branch.

k12th, 7 декабря 2012 в 01:52 # ↑ 0

Расскажите, если не трудно?

alexanderzaytsev, 7 декабря 2012 в 12:48 (комментарий был изменён) # ↑ 0

Я не большой эксперт в filter-branch, в основном нахожу ответы на SO. Окончания фиксятся примерно так:
stackoverflow.com/a/1060828/259946 (лично не проверял)

k12th, 7 декабря 2012 в 13:26 # ↑ 0

Круто, спасибо большое!

f0y, 6 декабря 2012 в 19:03 # ↑ 0

Она не всегда спасает. Вот тут сказано почему stackoverflow.com/questions/2333424/distributing-git-configuration-with-the-code/2354278#2354278

Из собственного опыта скажу что git blame <имя_файла> на виндовой машине будет говорит, что изменения not commited. Поскольку в репозитории юниксовые, а на винде виндовые и он думает что файл изменился. И поэтому надо прописывать git blame HEAD <имя_файла>. Также при мерджах возникает конфликт, но после вызова mergetool сразу говорит, что разрешён. Почему так еще не разобрался…

conf, 6 декабря 2012 в 13:50 # 0

Отличная статья! Все расписано очень внятно и доходчиво, спасибо.

FrenzyKryger, 6 декабря 2012 в 14:43 (комментарий был изменён) # +2

наконец-то есть что-то годное про rebase на русском чтобы дать почитать вместо того чтобы объяснять снова :)

alexanderzaytsev, 6 декабря 2012 в 14:56 # +1

Для чего перебазировать фиче-бранчи? Только ради эстетического удовольствия или есть еще какой-то в этом сакральный смысл?

bladeofsteel, 6 декабря 2012 в 15:11 # ↑ 0

Что бы сливаться без конфликтов, простым FF

alexanderzaytsev, 6 декабря 2012 в 15:19 # ↑ +2

Вы лукавите.

Допустим у вас в master есть изменения файла A и в ветке feature-1 есть N изменений этого же файла, и причем где-то в ближе к ветвлению есть конфликтующие изменения, то конфликты вы будете разрешать для этого файла >=N раз. Для сравнения с merge — 1 раз. А если не дай бог вы ошибетесь при разрешении конфликтов где-то в начале…

Для того чтобы сливаться без конфликтов есть rerere.

Dr_Logic, 6 декабря 2012 в 15:26 # ↑ +1

Если вы несколько раз меняете один и тот же файл, одно и то же место — то такие коммиты лучше squash-ить.
Да проблема о которой вы говорите, есть и решается она rerere и squash-ем. Merge в данном случае позволяет слиться быстрее — это факт.

alexanderzaytsev, 6 декабря 2012 в 15:32 (комментарий был изменён) # ↑ 0

Не обязательно в моей ветке менять одни и те же строки, достаточно в разных ветках эти строки поменять. В моем случае (я .NET программист) очень большую проблему представляют файлы проектов Visual Studio, т.к. студия добавляет файлы практическое случайное место в проект. А решать конфликты в .proj файлах ну очень неприятно.

Т.е. я не хочу сквошить коммиты из-за какого-то там файла проекта, который по большому счету вообще не нужен. И rerere на нем плохо работает:(

Dr_Logic, 6 декабря 2012 в 21:31 # ↑ 0

Да, тогда конечно squash тут не поможет не думал о такое ситуации.

bladeofsteel, 6 декабря 2012 в 15:29 # ↑ 0

> вы будете разрешать для этого файла >=N раз

Почему больше N? Разве не 1..N?
Действительно, может оказаться так, что конфликты в одном и том же месте придется разрешать несколько раз. Но есть большая вероятность, что они окажутся проще, чем при merge.

alexanderzaytsev, 6 декабря 2012 в 15:32 # ↑ 0

Да, я хотел написать меньше конечно же.

amosk, 6 декабря 2012 в 15:34 # ↑ +1

Без мержей, а не конфликтов.
Конфликты могут быть в обоих случаях.

bladeofsteel, 6 декабря 2012 в 15:35 # ↑ 0

Если Вы отребейзили фиче-ветку от мастера, то она вливается в мастер без конфликтов. Я писал об этом

bladeofsteel, 6 декабря 2012 в 15:37 # ↑ 0

Но, по большому счету, основной смысл в ребейзе — поддерживать порядок в графе ревизий.

alexanderzaytsev, 6 декабря 2012 в 15:40 # ↑ 0

Эстетический порядок? А зачем?

bladeofsteel, 6 декабря 2012 в 15:42 # ↑ 0

когда одновременно есть десяток-два веток становится сложно в этом ориентироваться. И не всегда все ветки вливаются в мастер, периодически фичи вливаются в фичи, особенно если над одной большой задачей работает несколько разработчиков.

Dr_Logic, 6 декабря 2012 в 15:46 (комментарий был изменён) # ↑ +3

Я бы не сказал, что дело только в эстетике. Ретроспективный анализ кода. Подготовка кода к слиянию. На мой взгляд отребейзенная ветка гораздо удобнее для этого. Вообще ребейз — не замена merge — вы после ребейза все равно делаете merge по fast-forward или без него, rebase — способ подготовить ваши изменения к этой операции. Помните, что отребейзенная ветка содержит в себе самый актуальный код, так как будто вы только что ее создали и внесли свои изменения, это позволяет протестировать ваши изменения до слияния их в мастер на самой актуальной кодовой базе. Убедил?

alexanderzaytsev, 6 декабря 2012 в 16:15 # ↑ +2

Да, я ждал этого комментария. С ссылкой наголо martinfowler.com/bliki/SemanticConflict.html

Дело в том, что в случае rebase все коммиты из фиче-бранча, кроме последнего (иногда и он) теряют свой смысл, кроме как «посмотреть».
Т.е. теряется смысл контроля версий.

В исходной статье коммиты C' и D' будут в лучшем случае в немного некорректном состоянии. Т.е. на них нельзя будет откатиться (отсюда появилась потребность в no-ff мерджах). К тому же, возможно придется добавить коммит F, который будет разрешать семантические конфликты.

Dr_Logic, 6 декабря 2012 в 16:36 # ↑ +1

Большое спасибо за ссылку, к сожалею не могу сейчас почитать — отложил в закладки. Не пойму, почему на них нельзя откатиться? Можно. Код не будет работать? Да возможно. Семантический конфликт? Это за пределами системы контроля версий.

Теряется смысл контроля версий из-за семантического конфликта? А часто вы деплоите приложение из тематической ветки? :) Вы рассматриваете именно конфликтные ситуации — в случае если изменения в разных модулях например — то всего этого нет, так?

А семантический конфликт разве не возможен при обычном merge? Не было такого что после подливания мастера ваш код ломался? Ну там функцию в api выпили например в мастере. Rebase — физический уровень манипуляции коммитами, он не защищает и не провоцирует семантические конфликты.

То что теряется старая ветка легко поправимо — если после ребейза сломалась ветка, а вам надо срочно сделать демонстрацию заказчику — можно сделать git reset --hard на коммит E (на него указывает ORIG_HEAD и его можно увидеть в git log --all. Ребейз можно отложить. Я не говорю о том, что мол rebase — золотая пуля. Это просто инструмент который может быть полезен.

Если делать ребейз периодически то фикс семантики можно делать не только в конце — а деплоиться на промежуточные версии (на мой взгляд) бессмысленно. Смысл системы контроля версий на мой взгляд не теряется. И кстати а «посмотреть» разве не входит в задачи scm? Code review, например, провести вполне можно.

Все ваши утверждения верны, тут вопрос в отношении к этим фактам и стилю работы с кодом, а также задачами, которые в каждом конкретном случае вы ставите перед scm.

alexanderzaytsev, 6 декабря 2012 в 17:48 # ↑ +1

То что семантический конфликт за пределами контроля версий это понятно, то что их может не быть — это понятно. Но при ребейзе будут плодиться коммиты «fix after rebase», а мердж позволяет решить их перед коммитом (--no-commit).

Мы активно практиковали подход с ребейзом, когда использовали меркуриал, т.к. там картина с мерджами выглядит еще ужасней. Но в нем нет ORIG_HEAD. Как-то я целый день (8 часов) сидел и пытался запушить свои изменения и меня постоянно кто-нибудь опережал (над проектом работало около 15 человек). После каждой попытки появлялся коммит из разряда «fix after rebase».

Я не говорю деплоить промежуточные коммиты. К примеру такая ситуация: сделал ребейз, решил конфликты (физические и семантические). Пошел на код ревью, тебе там несколько коммитов ближе к HEAD запороли и сказали, что их нужно переделать по другому. Иду, откатываюсь на последний успешно проревьювенный коммит… опять правлю семантические конфликты (для этого коммита они могут быть другими).

burdakovd, 6 декабря 2012 в 19:58 (комментарий был изменён) # ↑ +1

Простите, но зачем зачем плодить коммиты «fix after rebase»? Коммит — это не что-то высеченное в камне.

Заребейзил, прогнал тесты, исправил семантические несоответствия, *каждое исправление закоммитил в отдельный коммит*.

После чего rebase -i, и fixup-им эти исправления в те места истории ветки, где они должны были быть. При этом все коммиты из истории окажутся в согласованном состоянии.

Dr_Logic, 6 декабря 2012 в 20:03 # ↑ 0

Почитал про fixup, спасибо за наводку!

alexanderzaytsev, 7 декабря 2012 в 01:21 # ↑ 0

>Заребейзил, прогнал тесты, исправил семантические несоответствия, каждое исправление закоммитил в отдельный коммит.

Вы предлагаете после каждого ребейзнутого коммита прогонять тесты? Допустим у меня в ветке N коммитов, как узнать к какому из них нужно прилепить каждое изменение? Если фикс затрагивает несколько коммитов? Т.е. сделать в N раз больше работы и стресса? Т.к. каждый конфликт — это большой стресс.

Ладно, fixup — это хорошо. Но как объяснить это джуниору?

>Коммит — это не что-то высеченное в камне.

В git — да, в Hg — нет:(

develop7, 7 декабря 2012 в 01:35 # ↑ 0

> в Hg — нет:(

а как же у вас RebaseExtension работал тогда?

kosmonaFFFt, 7 декабря 2012 в 09:39 # ↑ 0

Все нормальные плагины к Mercurial, нацеленные на изменение истории коммитов работают только с той историей, которая никуда из локальной машины не ушла… После pull'а с твоей машины кем-нибудь или push на другую менять историю нельзя…

develop7, 7 декабря 2012 в 10:41 # ↑ 0

Да уж я-то знаю.

> Все нормальные плагины к Mercurial, нацеленные на изменение истории коммитов работают только с той историей, которая никуда из локальной машины не ушла…

а если точнее — то с коммитами в draft phase. Но и этого мне кажется вполне достаточно.

> После pull'а с твоей машины кем-нибудь или push на другую менять историю нельзя…

Ну почему нельзя. Пострипал ненужный кусок истории на всех машинах, на которые она успела расползтись — и готово.

alexanderzaytsev, 7 декабря 2012 в 11:37 # ↑ **–1**

Удачи со стрипом на bitbucket и прочих репо-хостингах… правильнее, на мой взгляд, делать мердж с отбросом изменений.

develop7, 7 декабря 2012 в 11:54 # ↑ **+1**

ну как раз на BB strip очень даже имеется.

alexanderzaytsev, 7 декабря 2012 в 12:35 # ↑ **0**

Спасибо, не знал, когда пользовался Hg там — еще не было. Сейчас только git.

Отвечу наперед, почему не GitHub — потому частные репозитории бесплатно.

alexanderzaytsev, 7 декабря 2012 в 11:35 # ↑ **0**

Кроме того, форс пуш не перезатирает ветку, а создает дополнительную голову, что для тех, кто хочет забрать изменения (особенно если включен аналог git pull --rebase) выливается в сплошной факап.

Нашли 3 способа решения проблемы с изменением уже запушеной истории:
— во всех репозиториях, включая оббщий делается стрип (применимо только, если вы хостите мастер-репу самостоятельно)
— делать мердж двух голов default с отбросом изменений из одной головы
— перейти на git

develop7, 7 декабря 2012 в 13:08 # ↑ **0**

> форс пуш не перезатирает ветку, а создает дополнительную голову

ну да, получается другой коммит с тем же предком

> 3 способа решения проблемы с изменением уже запушеной истории

Я ещё раз извиняюсь, но зачем именно править именно расшаренную историю? И что мешает править историю перед тем, как шарить?

alexanderzaytsev, 7 декабря 2012 в 13:28 # ↑ **0**

Я тут про Hg рассуждал. В нем форс пуш работает иначе чем в git. Создается дополнительная голова и у вас становиться 2 ветки default, если у кого-то стоит On Update: Rebase или On Update: Merge то он получит факап при апдейте.

Править апстрим нужно к примеру, после таких факапов, если их случайно запушили, если в апстрим попали битые коммиты, которые крашат клиент (я не знаю как такое случается, но бывало). Да мало ли что случилось?

develop7, 7 декабря 2012 в 14:19 # ↑ **0**

> Создается дополнительная голова и у вас становиться 2 ветки default

не совсем. выпинываются новые коммиты, а старые не стираются — поэтому и получается несколько голов в ветке.

> Править апстрим нужно к примеру, после таких факапов, если их случайно запушили

случайно скомандовали push -f? ну надо же.

alexanderzaytsev, 7 декабря 2012 в 14:29 # ↑ **0**

В некоторых клиентах, к примеру TortoiseHG есть такая галочка, а пушить из консоли не все умеют\хотят, поэтому когда нужно сделать push -f (к примеру, в старых версиях не было --new-branch для создания новой ветки, только через -f) ставили галочку, а потом ее забывали\ленились убрать.

develop7, 7 декабря 2012 в 16:55 # ↑ **0**

хм. ключику-то уже почти два с половиной года

alexanderzaytsev, 7 декабря 2012 в 17:01 (комментарий был изменён) # ↑ **0**

Я 2 года как перешел полностью на git.

kosmonaFFFt, 7 декабря 2012 в 13:20 # ↑ **0**

Способ 4 — не править историю…

burdakovd, 7 декабря 2012 в 01:44 # ↑ **+1**

> Вы предлагаете после каждого ребейзнутого коммита прогонять тесты?
Нет, тесты прогоняем один раз после rebase. Ну да, есть конечно небольшой шанс, что fixup я сделаю не туда, и какой-то из промежуточных коммитов окажется несогласованным. Но ведь есть Code Review (и log --stat/log -p), где будет видно, если какой-то из коммитов содержит не относящийся к нему фикс.

> Допустим у меня в ветке N коммитов, как узнать к какому из них нужно прилепить каждое изменение? Если фикс затрагивает несколько коммитов?
Может у меня пока не было сложных случаев, но всегда было понятно, к какому из коммитов локальной ветки нужно применить фикс. Ведь даже в локальной ветке коммиты имеют понятный commit message, видя который можно понять, относится фикс к данному коммиту или нет. Если фикс затрагивает несколько коммитов, то либо эти несколько коммитов должны были ранее быть объединены в один, либо фикс нужно было не одним коммитом делать, а тоже несколькими. На худой конец (если в ветке много несвязанных коммитов, все используют функцию foo и в апстриме она была переименована в bar) filter-branch спасёт, хотя мне почти не приходилось к нему прибегать.

> Ладно, fixup — это хорошо. Но как объяснить это джуниору?
Я сам джуниор. Понадобился в работе git — пошёл, прочитал несколько идеологических статей чтобы понимать «что такое хорошо, и что такое плохо». Когда нужно какую-то конкретную задачу решить — используем поиск и читаем доки по конкретной команде. В чем проблема-то?

alexanderzaytsev, 7 декабря 2012 в 07:50 # ↑ 0

>Я сам джуниор. Понадобился в работе git — пошёл, прочитал

Был у нас паренек, он уволился через месяц, сказал что такой информационной нагрузки не выдерживает)

А вообще тут все дело в мотивации — если человек сам не захочет научиться git-у (или чему другому) — он не научится, хоть ты палкой его бей.

develop7, 7 декабря 2012 в 11:10 # ↑ +1

> сказал что такой информационной нагрузки не выдерживает

Я извиняюсь, но конкретно git требует незаслуженно много внимания пользователя, в отличие от аналогов.

develop7, 6 декабря 2012 в 22:20 # ↑ 0

> Как-то я целый день (8 часов) сидел и пытался запушить свои изменения и меня постоянно кто-нибудь опережал (над проектом работало около 15 человек)

(голосом Олега Табакова) Это ж ~~бубль-гум~~subversion!

alexanderzaytsev, 7 декабря 2012 в 01:22 # ↑ 0

К сожалению это был Hg.

A1lfeG, 7 декабря 2012 в 11:11 # ↑ 0

> . Но в нем нет ORIG_HEAD. Как-то я целый день (8 часов) сидел и пытался запушить свои изменения и меня постоянно кто-нибудь опережал (над проектом работало около 15 человек).

15 человек работающие в одной ветке? Как тут гит поможет?
Мне действительно инетересно.

alexanderzaytsev, 7 декабря 2012 в 11:41 # ↑ 0

Вы не поняли.

Каждый работает в своей ветке, а потом сливает в master (default для Hg) когда фича готова, просто так совпало что я свою фичу закончил практически одновременно с еще несколькими членами команды, которые оказались шустрее меня.

amosk, 6 декабря 2012 в 15:38 # ↑ 0

Я понимаю, что вы имели в виду.
Но у новичков от вашего утверждения может создаться ложное впечатление что rebase позволяет избежать конфликтов.

bladeofsteel, 6 декабря 2012 в 15:40 # ↑ 0

Пожалуй, да — так можно прочитать

alexanderzaytsev, 6 декабря 2012 в 15:43 # ↑ 0

Чтобы «отребейзить» фиче-бранч от мастера нужно все те же конфликты решить, разве нет? А у меня бывали ситуации, когда при простом мердже в мастер конфликтов нет, а при ребейзе… есть.

bladeofsteel, 6 декабря 2012 в 15:50 # ↑ 0

Решать надо.

Не знаю как у Вас построен процесс, а у нас в мастер вливает только тимлид. Владелец ветки перед передачей на слияние ребейзит ветку от мастера и решает конфликты, если они есть. Тимлиду же остается только влить изменения (после ревью). Конфликт решить проще разработчику, нежели тимлиду.

Agent_Smith, 6 декабря 2012 в 15:55 # ↑ 0

К тому-же после ребейса еще полезно будет прогнать тесты на актуальной кодовой базе, и уже потом кидать пулл реквест.

alexanderzaytsev, 6 декабря 2012 в 16:36 # ↑ 0

У вас тимлиду больше заняться нечем?

Как быть с ситуацией, когда исполнитель сделал ребейз, отправил код на ревью, его проревьювили и не форварднули по каким-либо причинам (бизнес сказал, что фичу нужно отложить, тимлид отвлекся и забыл, и т.п.). Тем временем мастер поплнился другими коммитами. Исполнителю опять нужно идти и ребазировать-решать конфликты, переключаться с текущих задач? Потом тимлиду *опять* нужно делать ревью, и круг повторяется.

Dr_Logic, 6 декабря 2012 в 16:38 # ↑ 0

>> Исполнителю опять нужно идти и ребазировать-решать конфликты
А при мердже этого делать разве не надо в такой ситуации? :) Тут проблема не merge или rebase — проблема в самой ситуации. Код устаревает постоянно пока не влит в мастер. После вливания в мастер код заставляет устаревать другой код, еще не влитый в мастер. Это жизнь.

alexanderzaytsev, 6 декабря 2012 в 16:47 # ↑ 0

Ну как-бы вы сами ответили на свой вопрос. Мердж почти атомарная операция: решил конфликты, закоммитил, запушил.

bladeofsteel, 6 декабря 2012 в 16:51 # ↑ 0

Слияние отложенной фичи может быть весьма проблематичным

bladeofsteel, 6 декабря 2012 в 16:49 # ↑ 0

Описываемая Вами ситуация для нашей команды экстраординарная (если для Вас это норма — искренне Вам сочувствую). Поэтому накладные расходы на повторный ребейз и ревью не нами учитываются (к тому же они весьма вероятно будут значительно меньше).

alexanderzaytsev, 6 декабря 2012 в 16:56 # ↑ 0

Они будут не меньше, а все те же 1..N на каждый конфликтный файл.

Я привел 2 вполне жизненных примера, сочувствие тут не уместно. Мы все живем в реальном мире.

bladeofsteel, 6 декабря 2012 в 17:01 # ↑ 0

> Они будут не меньше, а все те же 1..N на каждый конфликтный файл.

Это будут уже другие конфликты. И высока вероятность, что их вообще не будет.
Я не утверждаю, что затраты **всегда** меньше, я говорю, что они **с большой вероятностью** будут меньше.

Похоже, что у нас слишком разный паттерн использования гита и продолжать спор смысла нет

alexanderzaytsev, 6 декабря 2012 в 17:16 # ↑ 0

У меня нет «паттерна» использования гита, я его использую так, как оправданно в конкретной ситуации.

Я просто хочу донести до вас, что почти с каждым ребейзом у вас будут появляться коммиты «fix after rebase».

bladeofsteel, 6 декабря 2012 в 17:20 # ↑ 0

за полтора года работы по такой схеме не было ни одного такого комита. Видимо нам везет. Или что-то делаем не так ;)

alexanderzaytsev, 6 декабря 2012 в 17:33 # ↑ +2

Одно из двух: либо вы один на проекте, либо каждый работает над своей изолированной частью. Что в принципе одно и то же. Либо вы опять лукавите.

У вас после ребейза никогда не было семантических конфликтов? Не верю.

Самое простое что пришло на ум:
Вы пишите код, вызываете какую-то функцию.

В это время злой Вася Пупкин обнаруживает, что эту функцию никто не использует кроме него и решает ее переименовать или отрефакторить. Затем ребазирует, успешно проходит code review, его изменения попадают в мастер.

Вы заканчиваете свою фичу, рабазируете и… «fix after rebase»

bladeofsteel, 6 декабря 2012 в 17:42 # ↑ +1

Команда у нас действительно не большая, но над одной задачей бывает работают несколько разработчиков.

О таких калечащих изменениях обычно сообщается всем и заранее. Все приблизительно в курсе, кто чем занимается, поэтому если есть необходимость что-то поменять, то об этом сообщают людям, которых это может затронуть.

Не понятно, чем в данном случае ребейз от мержа будет отличаться? Только тем, что семантический конфликт будет зарыт внутри мерж-комита, а не идти отдельным?

alexanderzaytsev, 6 декабря 2012 в 17:50 # ↑ 0

>Только тем, что семантический конфликт будет зарыт внутри мерж-комита, а не идти отдельным

Да, и решить его нужно будет 1 раз.

alexanderzaytsev, 6 декабря 2012 в 16:50 # ↑ 0

Забыл. Утверждение, что владельцу ветки проще решить конфликты чем тимлиду верно только от части. Исполнитель отвечает за свои изменения, тимлид за чужие. Но голова у тимлида не резиновая и он позовёт Васю Пупкина, который сделал те изменения, с которыми конфликтуют изменения исполнителя.

bladeofsteel, 6 декабря 2012 в 16:55 # ↑ 0

Вот именно, ситуация когда разработчик не знает как правильно решить конфликт бывает гораздо реже (голова у тимлида не резиновая, тут Вы верно заметили).
Для решения конфликта тимлид наверняка привлечет разработчика ветки. Так почему бы разработчику самому не решать конфликты, без привлечения тимлида?

alexanderzaytsev, 6 декабря 2012 в 16:58 # ↑ +1

Потому что разработчику придется самому привлекать другого члена команды через тимлида или непосредственно, т.к. он не отвечает за чужие изменения, сделанные пока он выполнял свою работу.

alexanderzaytsev, 6 декабря 2012 в 17:52 # ↑ 0

Кстати, мне, как человеку, который часто ревьювит код проще смотреть в одном коммите, чем высматривать построчные изменения в 50 коммитах. Так видно всю картину целиком.

bladeofsteel, 6 декабря 2012 в 17:56 # ↑ 0

git diff делает точно так же — смотрим различия между ветками целиком

youROCK, 6 декабря 2012 в 17:55 (комментарий был изменён) # ↑ 0

> Владелец ветки перед передачей на слияние ребейзит ветку от мастера и решает конфликты, если они есть.

~~А теперь нужно откатить одну из 15 веток, которая таким образом была замержена, потому что она вызывает проблемы :). Как вы будете это делать с таким подходом? Ведь у вас линейная история.~~

Прошу прощения, погорячился, «отмержить» ветку тоже достаточно легко в случае линейной истории — нужно просто удалить диапазон коммитов из ветки. Согласен.

bladeofsteel, 6 декабря 2012 в 17:57 # ↑ 0

Что попало в мастер из него не выпиливается. Появилась проблема — делай фикс

Dr_Logic, 6 декабря 2012 в 15:14 # ↑ +2

Да и когда ветка долго живет в ней копятся мердж коммиты (чтобы держать кодовую базу в ветке актуальной) — при ребейзе про состояние до ребейза можно забыть, как будто вы только что обранчевались и накоммитили в новую ветку.

alexanderzaytsev, 6 декабря 2012 в 15:22 (комментарий был изменён) # ↑ 0

А откуда в фиче-бранче копятся мерджи из master?

Dr_Logic, 6 декабря 2012 в 15:28 (комментарий был изменён) # ↑ +2

А вы не подливаете себе мастер, если ветка долго живет? Потом все разом мерджите? Это сложно бывает, я бы рекомендовал раз в несколько дней подмерджить мастер, чтобы потом с ума не сойти.

alexanderzaytsev, 6 декабря 2012 в 15:38 # ↑ +2

Стараюсь этого всячески избегать.

Dr_Logic, 6 декабря 2012 в 15:47 (комментарий был изменён) # ↑ +1

Почему? Вам не нужен актуальный код в вашей ветке? А если окружение поменялось и ваша ветка в нем попросту не работает?

alexanderzaytsev, 6 декабря 2012 в 16:17 # ↑ –1

Эм, это как так? Как моя ветка может сломаться, если ее *никто* не трогал?

youROCK, 6 декабря 2012 в 16:29 # ↑ 0

Если у вас общее окружение (т.е. вы не одни на разработческом сервере :))), то любой рефакторинг, который затрагивает смену этого окружения, сломает вашу ветку. Например, API какого-нибудь внутреннего сервиса сменился. В master есть нужные изменения, а в вашей ветке — нет.

alexanderzaytsev, 6 декабря 2012 в 16:43 # ↑ –1

Т.е. у вас есть фиче-бранчи и общее окружение. Печаль.

Хорошо, в ветке Васи Пупкина есть изменения которые сломали ваше окружение, но его ветка еще не готова быть ни в вашей ветке, ни в мастере. Работа встала. Печаль.

То что вы говорите для меня вообще невероятная ситуация.

Мой рабочий компьютер и есть «разработческий сервер».

Dr_Logic, 6 декабря 2012 в 16:48 # ↑ 0

И пока вы работаете в ветке вы не переключаетесь на другую ветку? Или все-таки у вас БД столько же сколько веток (или она редко меняется, или изменения непротиворечащие). А зачем вам ветки в таком случае вообще? :)

alexanderzaytsev, 6 декабря 2012 в 17:08 # ↑ 0

А зачем переключаться?

— Все зависит от проекта: есть окружение, которое тесно связано с версией (к примеру схема в RDBM), есть ресурсы неизменные.

Если говорить про базы, то есть бэкап какой-то версии мастера и есть мигратор, который может обновить схему до нужной версии и добавить нужные данные.

youROCK, 6 декабря 2012 в 16:53 # ↑ 0

Ну, мы работаем в других условиях. У нас слишком накладно поднимать на каждом компьютере разработчика и держать в актуальном состоянии по копии всех нужных баз и, что более важно, сишных сервисов, в том числе мемкешей.

alexanderzaytsev, 6 декабря 2012 в 17:13 # ↑ 0

Базы не обязательно должны быть актуальные, нужна схема базы, которая соответствует разрабатываемой версии. Это можно получить быстро из бэкапа той версии, от которой сделана ветка и мигратора.

Мемкэш не особо нуждается в версионировании и вполне себе может быть общим.

Dr_Logic, 6 декабря 2012 в 16:39 # ↑ 0

Да не обязательно апи, схема бд может поменяться например, или у вас для каждой ветки своя БД развернута?

defuz, 6 декабря 2012 в 18:07 # ↑ +1

Для меня главный смысл делать rebase — резолвить ситуации, когда в feature-branch нужно забрать обновленный код из мастера (например, изменилось апи, которое использует фича). Можно конечно сделать merge master в feature, но это создает такую кашу в репозитории, что иногда очень легко запутаться в происходящем. С ребейзом все куда проще.

alexanderzaytsev, 6 декабря 2012 в 18:21 # ↑ 0

Вот это подход — правильный. Ребазировать всегда, чтоб сделать линейную историю — нет.

bladeofsteel, 6 декабря 2012 в 21:18 # ↑ +1

Так, вроде, никто про линейную историю не говорил

smaant, 1 ноября 2013 в 19:50 # ↑ 0

Простите, что откапываю стюардессу, но тема показалась интересной.

Но ведь если с помощью rebase «резолвятся ситуации, когда в feature-branch нужно забрать обновленный код из мастера» с легкостью можно оказаться в ситуации когда все коммиты в feature-branch будут содержать семантический конфликт. Например если в мастере изменили название какого-то метода, который используется в бранче. Т.е. имхо использовать rebase в этой ситуации тоже не очень безопасно.

Dr_Logic, 2 ноября 2013 в 00:07 # ↑ 0

Я думаю, вам будет интересно почитать эту ветку обсуждений habrahabr.ru/post/161009/?reply_to=6933652#comment_5547341 :)

smaant, 2 ноября 2013 в 11:49 # ↑ 0

Спасибо, я уже прочитал все комменты конечно. Из этого обсуждения собственно и узнал о семантических конфликтах.

0

youROCK, 6 декабря 2012 в 15:21 #

Споры по поводу того, что лучше — merge или rebase, как мне кажется, не имеют смысла, потому что rebase является надстройкой над merge и, исходя из этого, не может приводить к меньшему количеству конфликтов, чем сам merge :).

С другой стороны, rebase позволяет переписать историю коммитов, сделав её более ровной.

alexanderzaytsev, 6 декабря 2012 в 15:27 # ↑ 0

> потому что rebase является надстройкой над merge

Что вы имеете ввиду?!

> исходя из этого, не может приводить к меньшему количеству конфликтов

Я написал в другой ветке, что rebase ведет к большему количеству разрешений конфликтов. Хотя сами конфликты становятся меньше.

> С другой стороны, rebase позволяет переписать историю коммитов, сделав её более ровной.

Дак я и спрашиваю — какой в этом смысл кроме псевдо-эстетического?

Dr_Logic, 6 декабря 2012 в 15:29 # ↑ +3

rebase не является надстройкой над merge, он является надстройкой (а точнее автоматизацией) cherry-pick-ов

alexanderzaytsev, 6 декабря 2012 в 15:46 # ↑ 0

К сожалению иногда пикнуть несколько коммитов вручную намного легче, чем через такую «автоматизацию»

Dr_Logic, 6 декабря 2012 в 15:51 # ↑ +1

В таком случае надо их просто пикнуть :) Инструменты созданы чтобы упростить нашу жизнь, а не усложнить. Стоит пользоваться самым подходящим — из пушки по воробьям стрелять никто не заставляет :)

alexanderzaytsev, 6 декабря 2012 в 16:19 # ↑ 0

Иногда сложно *заранее* понять, что будет проще использовать, если в вашей фиче больше чем 2 коммита.

youROCK, 6 декабря 2012 в 15:54 (комментарий был изменён) # ↑ 0

Ну а cherry-pick, в свою очередь, таки является надстройкой над merge :), хоть это и не так легко найти в коде гита.

Заодно прикольно было узнать, что один и тот же код делает как git revert, так и git cherry-pick (хотя, если подумать, это довольно логично, т.к. revert является просто cherry-pick от коммита, в котором "+" заменен на "-" :))

```
/* sequencer.c */
...
static int do_pick_commit(struct commit *commit, struct replay_opts *opts)
{
    ...
    if (!opts->strategy || !strcmp(opts->strategy, "recursive") || opts->action == REPLAY_REVERT) {
        res = do_recursive_merge(base, next, base_label, next_label,
                     head, &msgbuf, opts);
        write_message(&msgbuf, defmsg);
    } else {
        ...
        res = try_merge_command(opts->strategy, opts->xopts_nr, opts->xopts,
                    common, sha1_to_hex(head), remotes);
        ...
    }
....
}
```

Dr_Logic, 6 декабря 2012 в 16:05 # ↑ +1

Сорри, не убедили. Вырванный из контекста кусок кода мне, к сожалению, ничего не доказывает. do_recursive_merge — просто наложение изменений на дерево проекта (наложение диффа), про try_merge_command — не могу сказать про что, некогда ковырять. Я говорил о merge как о команде гита, которая создает коммит с двумя родителями. По вашей логике patch — тоже merge. Если вы это и имели ввиду — то ок, спор чисто терминологический и мы друг друга просто недопоняли.

Если вы считаете что rebase или cherry-pick можно разложить на несколько мерджей (в моей терминологии), попробуйте это в двух словах сделать.

youROCK, 6 декабря 2012 в 16:21 (комментарий был изменён) # ↑ 0

try_merge_command вызывает git merge с выбранной стратегией (см. спойлер).

Скрытый текст

Но да, вы правы, для того, чтобы rebase использовал свой же merge, нужно явно выбрать "--merge", когда делается rebase.

А принцип использования мержа очень простой (для git rebase onto A из ветки B):

1. находим merge-base между A и B = C (точка расхождения истории коммитов)
2. i = 1
3. мержим коммит номер i (считая из точки C в направлении B) в ветку A (находясь в ветке A, делаем git merge <sha1>)
4. повторяем до тех пор, пока не закончатся коммиты из B

Да, rebase --interactive и squash коммитов так сделать нельзя, но перенос коммитов — можно :).

 **Dr_Logic,** 6 декабря 2012 в 16:46 # ↑ 0

Опять вы говорите о merge как об операции наложения изменений. Git merge тоже основан на merge. Git stash apply тоже основан на merge. Только споры обычно идут операции git merge, а точнее о том, стоит ли перед ней делать git rebase или нет. А еще о том, как подливать кодовую базу в тематическую ветку — через перебазирование самой ветки или через слияние изменений. Спасибо за интересную беседу anyway. У меня до ковыряния в сорцах гита еще не доходило.

 **youROCK,** 6 декабря 2012 в 16:55 # ↑ 0

Моя мысль лишь в том, что бОльшую часть операций гита можно свести к мержу, поэтому устраивать холиворы из-за надстроек над мержем как-то странно :). Я ещё понимаю холиворы между разными DVCS, в которых по-разному хранятся данные и устроен merge, но чтобы между merge и rebase — это уже действительно Linux-way :).

 **youROCK,** 6 декабря 2012 в 16:57 # ↑ 0

> У меня до ковыряния в сорцах гита еще не доходило.
К сожалению, некоторые вещи, которые касаются внутреннего устройства гита, нигде толком не описаны, поэтому приходится лезть в исходный код, чтобы понять, что же там на самом деле происходит.

 **Dr_Logic,** 6 декабря 2012 в 15:37 # ↑ +2

Цель ребейза не уменьшить количество конфликтов, а уменьшить количество мержей в истории. Так что сравнивать по количеству/качеству конфликтов — не имеет смысла — конфликты они в вашем коде, а не в операции, которой вы его сливаете — просто при мердже вы разруливаете конфликт для конечных версий в ветках, а при rebase конфликт каждого изменения (коммит-а).

 **youROCK,** 6 декабря 2012 в 15:56 # ↑ 0

Цель rebase понятна. Я говорил о том, что спор «rebase против merge» не имеет смысла, поскольку rebase это надстройка над merge, просто очень навороченная.

 **nazarpc,** 6 декабря 2012 в 15:48 # 0

Меня одного путает направление стрелок?

 **1nd1go,** 6 декабря 2012 в 15:53 # ↑ 0

Ооо, вы украли мой коммент :) Я тоже не понял, почему стрелки в обратную сторону!

 **Assorium,** 6 декабря 2012 в 15:58 # ↑ 0

Так устроена система хранения в GIT. Коммит указывает на родителя/родителей.
Хотите познать GIT? GIT.pro Вам в помощь!

 **Dr_Logic,** 6 декабря 2012 в 16:08 # ↑ +2

Прочитайте пожалуйста мой коммент. Очень надеюсь, что он поможет понять почему стрелки нарисованы именно так, а не иначе.

 **1nd1go,** 6 декабря 2012 в 16:19 # ↑ 0

Да, спасибо. Как вы правильно заметили, «на github стрелки рисуют в другую сторону.», наверное все же так понятнее. В школе сначала тоже «ускорение» пишут как «a», а не dv/dt.

 **Dr_Logic,** 6 декабря 2012 в 15:59 # ↑ +1

Направление от потомка к родителю соответствует физике процесса. Дело в том что в коммите B например хранится указатель на коммит A (хэш родительского коммита). Отпугивает видимо не только вас, так как на github стрелки рисуют в другую сторону. Однако в литературе например рисуют также. Нужно стрелку воспринимать как указатель, а не как направление действия.

 **nazarpc,** 6 декабря 2012 в 16:05 # ↑ 0

Мне почему-то хронологическое направление кажется более естественным.
Сначала было одно, затем другое, потом третье.

 **Dr_Logic,** 6 декабря 2012 в 16:07 # ↑ +2

Это если трактовать стрелку как действие. Тут стрелки отображают структуру данных (ну вы наверняка в универе рисовали односвязные списки, можно считать, что это они).

 **Emiya,** 6 декабря 2012 в 20:17 (комментарий был изменён) # 0

Использую и merge и rebase.

Удобство merge в «маленьких» commit-ах, каждый из которых не обязан иметь законченное решение. Получает, что ветка это удобная история изменений, каждый merge-commit это законченное решение. Ни кто, ни кому не мешает при разработке в группе.

Удобство rebase в «прямолинейности» ветки, но каждый commit это законченное решение, или нужно будет создавать теги. Не очень удобен, поскольку постоянно придется следить кто какие commit-ы уже сделал,

 **Dr_Logic,** 6 декабря 2012 в 23:01 # ↑ 0

Что мешает ребейзенную ветку влить в мастер merge-ем с опцией --no-ff? Выйдет тоже самое, не?

 **Splurov,** 6 декабря 2012 в 23:49 # +1

Как быть в таком случае:
Есть ветки master, feature-1 (основа — master), feature-2 (основа — master). В каждую из веток регулярно коммитят.
Возникает задача параллельно делать feature-3, которая будет включать себя изменения из feature-1 и feature-2 (и соответственно изменения из master).
При этом по ходу разработки в feature-3 нужно регулярно подтягивать изменения feature-1 и feature-2 (которые меняются и регулярно ребейзятся на master).
Если в feature-3 для получения изменения из 1 и 2 я буду использовать rebase, то коммиты постоянно будут дублироваться (что логично, ведь git не будет знать о том, что коммиты в feature-1 и feature-2 «те же самые»).
Если использовать merge, то история загрязнится мердж-коммитами и почти невозможно будет получить «чистую» история по feature-3.

Как вариант, для получения изменений делать squash всех коммитов в feature-3 и уже потом rebase на ветки, но теряется история, что не удобно.
Может быть я упускаю какое-то простое очевидное решение?

 **Dr_Logic,** 7 декабря 2012 в 00:56 # ↑ 0

Реально сложная задача, может она неправильно поставлена? Почему все три фичи не могут делаться в одной ветке если они так тесно взаимосвязаны?

 **Splurov,** 7 декабря 2012 в 00:58 # ↑ 0

Тестироваться и выпускаться (мерджиться в мастер) будут отдельно.

 **Dr_Logic,** 7 декабря 2012 в 01:08 # ↑ 0

Можно ли разбить фичи 1 и 2 на этапы? Например после завершения определенной работы в ветке 1 и 2 они готовы к вливанию в 3? В каком порядке фичи будут мерджиться в мастер заранее известно или по готовности? Может вам стоит пилить общий для всех этих 3 веток функционал в базовой ветке, относительно которой ребейзить все эти три фиче-ветки? И там уже тестировать перед вливанием в мастер?

Мне что-то то кажется что между вашими тремя ветками и мастером должна быть промежуточная ветка в которую вы вливаете изменения из этих трех.

 **Splurov,** 7 декабря 2012 в 01:17 # ↑ 0

Если в очень упрощённом варианте: сделали 1 и 2, отдали в тестирование, начинаем делать 3, зависящую от 1 и 2. В 1 и 2 тестировщик нашёл баги, правим, 3 нужно обновить. И так несколько раз.
Как будут мержится известно, но 1 и 2 раньше. Предполагается, что к моменту передачи в тестирование 3 фичи 1 и 2 уже будут выпущены (или по крайней мере протестированы).
Всё в одной не удобно — сложнее тестировать, сложнее планировать и следить за процессом разработки.

 **Dr_Logic,** 7 декабря 2012 в 01:17 # ↑ 0

Стойте, а код из ветки 1 в ветку 2 не может попадать? То есть изменения объединяются только в ветке 3?

 **Splurov,** 7 декабря 2012 в 01:18 (комментарий был изменён) # ↑ 0

Из 1 в 2 не может попадать. Объединяются только в 3, да.

 **Dr_Logic,** 7 декабря 2012 в 01:42 # ↑ +2

У вас тут есть одно внутреннее противоречие из-за которого все проблемы:
— ветки 1 и 2 должны вестись изолировано
— ветка 3 должна содержать объединенные изменения из ветки 1 и 2.
Отсюда следует, что эти изменения не могут храниться в одних и тех же коммитах. Два выхода:
1. Либо изменения приезжают в ветку 3 с помощью мерджа и храняться в мердж-коммитах — запутываем граф.
2. Либо изменения из веток 1 и 2 дублируются в других коммитах — в ветке три — их можно переносить туда cherry-pick-ом, но тогда готовьтесь к конфликтам и дублированию пикнутых коммитов при интеграции ветки 3 в мастер, в который уже проинтегрированы ветки 1 и 2. Ну и черрипик не слишком быстрая операция так как предполагает много ручной работы.
3. Либо вам надо отказаться от полной изолированности ветки 1 и 2, то есть когда ветки 1 и 2 готовы к интеграции в ветку 3 они склеиваются ребейзом до текущего состояния (можно пометить его веткой release) и далее снова разрабатываются изолировано. Ветка 3 ребейзится на состояние release.

Третий вариант я не посчитал, так как он нарушает одно из входящих условий

 **Splurov,** 7 декабря 2012 в 11:36 # ↑ 0

На счёт варианта 2 подумаю, спасибо.

 **Dr_Logic,** 7 декабря 2012 в 01:54 # ↑ +1

А вот еще классный вариант придумал — не делать фичу 3 пока не сделаны первые две. Сложно писать код на такой зыбкой почве как две динамично меняющиеся кодовые базы совместимость которых тестится только у меня в бранче. Ну и да декомпозиция задач 1 и 2 на более мелкие может облегчить этот процесс за счет того, что каждый этап занимает меньше времени и по сути делается в отдельной ветке (может с тем же названием, может «новая» ветка — продолжение отребейзенной старой)

alexanderzaytsev, 7 декабря 2012 в 07:53 # ↑ 0

Теперь объясните это дяденьке, который деньги из большой пачки отслюнявливает.

А ситуация, которую описывает Splurov, к сожалению, жизненная, и мне встречалась.

Splurov, 7 декабря 2012 в 11:37 # ↑ 0

К сожалению, это невозможно.

burdakovd, 7 декабря 2012 в 01:59 # ↑ 0

Мне кажется если «в feature-3 нужно регулярно подтягивать изменения feature-1 и feature-2», то feature-1 и feature-2 не должны «регулярно ребейзятся на master».

В противном случае каким бы мы способом ни вливали изменения из feature-1, feature-2 в feature-3 — проблем не избежать. Rule of thumb: не ребэйзить ветку, если на неё кто-то ссылается (или имеет локальную копию, или если она есть в публичном репозитории). Хоть автор статьи и утверждает «здесь мы рассмотрели, как легко можно избежать этой проблемы с помощью git pull --rebase», но на самом деле проблемы так не решатся, и, как мне кажется могут быть решены вовсе: изменение истории слишком грязный трюк, чтобы делать его где-то кроме локальной ветки, которую никто не видел.

burdakovd, 7 декабря 2012 в 02:07 (комментарий был изменён) # ↑ +2

Если же предположим, что ветки 1 и 2 не ребейзятся, то всё более-менее просто.

Делаем вспомогательную ветку feature-3-base, куда периодически мержим изменения из веток 1 и 2 и master, и своих коммитов в неё не делаем. А ветку feature-3 периодически ребейзим на feature-3-base. Когда ветки 1 и 2 вмержатся в мастер, через git rebase --onto перенесем третью ветку с feature-3-base на мастер. Аналогичные действия можно сделать если в мастер вмержилась только первая или только вторая ветка.

В принципе, используя rebase --onto можно пытаться делать что-то похожее даже если первая и вторая ветки регулярно _ребэйзятся_ на мастер.

Если мерж первой, второй ветки и мастера нетривиален — то возникает проблема с тем, что мы его делаем несколько раз (сперва мержим чтобы использовать в feature3, а потом повторяем те же действия, когда будем отправлять первую и вторую ветку в мастер). rerere как-то поможет, но всё-равно неприятно.

Splurov, 7 декабря 2012 в 11:50 # ↑ 0

Интересный вариант и, похоже, решает проблему, спасибо.

Dr_Logic, 7 декабря 2012 в 02:26 (комментарий был изменён) # ↑ 0

Правда? Пока проблем не обнаружено. Со вторичными тематическими ветками не пробовал но шаренные ветки ребейзил. git pull --rebase вытягивает новую ветку и перемещает локальные коммиты наверх. Да и git help pull говорит:

> --rebase
> Rebase the current branch on top of the upstream branch after fetching. If there is a remote-tracking branch corresponding to the upstream branch and the **upstream branch was rebased**
> **since last fetched**, the rebase uses that information to avoid rebasing non-local changes.

Что намекает, что это допустимая ситуация. Возможно в более ранних версиях гита это был грязный трюк, но сейчас это вполне работает.

burdakovd, 7 декабря 2012 в 09:25 # ↑ 0

Не знал. Но это похоже на хак, а решает проблему только в очень частном случае. Хотел написать подробнее, но тут уже всё отлично написали.

burdakovd, 7 декабря 2012 в 09:30 # ↑ +1

Ну и когда мы пушим отребейзенную ветку: «git push origin feature --force» у нас есть race condition, ведь если кто-то запушил в эту ветку свой коммит за секунду до того как мы запушили с форсом, его коммит из репозитория пропадёт. После чего когда он сделает у себя git pull --rebase, этот коммит *наверно* пропадёт и из его локальной копии (т.к. его копия git не считает этот коммит локальным и не будет его ребэйзить).

m00t, 7 декабря 2012 в 09:37 # ↑ 0

+1, про race-condition я даже и не подумал, когда писал коммент! А ведь действительно, причем там не секунды синхронизации, далеко не секунды. Нужно обязательно перед каждым push --force делать тот же pull --rebase. Если забыл и затер изменения коллеги — будет много недовольных в лучшем случае. Т.е. добавляется еще одно административное ограничение кроме как запускать git pull --rebase — запускать его пере push, причем прямо перед самым им, чтобы минимизировать между ними время и, соответственно, вероятность race condition

alexanderzaytsev, 7 декабря 2012 в 11:44 # ↑ 0

Действительно, получится только минимизировать, а не исключить. Поэтому за push --force нужно отрубать руки.

m00t, 7 декабря 2012 в 12:11 # ↑ 0

Я бы не стал так категорично заявлять. Как стандартный workflow, push -f, конечно, сложно принять. Но как инструмент для применения иногда — почему бы и нет. Раз уж у нас есть такой крутой git pull --rebase

alexanderzaytsev, 7 декабря 2012 в 12:36 # ↑ 0

Ключевое слово тут «иногда».

Dr_Logic, 7 декабря 2012 в 14:11 # ↑ 0

Race condition фактически есть, но реально, это надуманная проблема — если я с кем-то работаю в одной ветке — значит я много общаюсь с ними, постоянно делюсь кодом, обсуждаю архитектуру. Ничего не мешает договориться о моменте ребейза и зафризить ветку — это конечно хак за пределами SCM, но мне при работе в общей ветке не приходилось сталкиваться с race condition. Насчет того пропадет ли «гоночный» коммит при ребейзе — надо поставить эксперимент — в теории выглядит именно так, но надо бы убедиться — поэкспериментирую на досуге.

alexanderzaytsev, 14 декабря 2012 в 01:06 # ↑ 0

Можно попробовать вот такое решение blog.caurea.org/2009/11/19/subtree-octopus-merge.html

Splurov, 15 декабря 2012 в 00:35 # ↑ 0

Спасибо, интересный вариант.

Glowfall, 7 декабря 2012 в 03:35 # 0

Хорошая статья… Вот бы еще убедить коллег придерживаться такого стиля, ведь «и так всё работает» :(

Dr_Logic, 7 декабря 2012 в 10:37 # ↑ 0

Дайте им ее почитать, вдруг они проникнутся. ;)

m00t, 7 декабря 2012 в 09:06 # +1

Это не очевидно, но git pull --rebase делает не просто git pull && git rebase. Как вы процитировали доки выше, он смотрит, не ребейзился ли апстрим, и если ребейзился, то он ребейзит только те изменения что добавили мы. Это вообще говоря, похоже не хак и корректно работает только в некоторых конкретных случаях: когда у нас есть один главный бранч и фичебранчи. Действительно тогда фичебранчи можно без проблем ребейзить, если все делают git pull --rebase при работе. Как только система становится сложнее (например, добавляется фичебранч, основанный на другом фичебранче, как тут), git pull --rebase уже не будет корректно работать (как, собственно, вы и написали в конце статьи). Подозреваю, что все станет еще хуже, как только у вас появятся релиз-бранчи, staging, хотфиксы в master. Если только не взять за правило ничего кроме фичебранчей не ребейзить.

Конечно, линейная история гита при использовании долгоживущих фичебранчей это круто. И я ни в коем случае не агитирую против такого подхода, потому что хистори гита это часть исходников и как и все другие исходники она должна быть чистой и понятной. (Я против фичебранчей в целом, но это не относится к теме этой статьи.) Но цена за такую линейную историю — грязные хаки в виде git pull --rebase, которые будут работать только в конкретных случаях. Т.е. как только у нас в репозитории произойдет что-то нестандартное, наш стандартный хак не сработает. Это должно быть как-то административно поставлено, что все используют только git pull --rebase, что тоже не очень хорошо, ИМХО.

Повторюсь, я не против такого подхода, если необходимость использования долгоиграющих (например, отдельно тестируемых) фичебранчей не ставится под сомнение, но нужно понимать границы применения, и цену такого подхода.

Dr_Logic, 7 декабря 2012 в 10:32 (комментарий был изменён) # ↑ +2

Git pull — rebase конечно не делает git pull и git rebase, так как git pull — это fetch & merge, а git pull --rebase — это fetch & rebase. Кроме того нужно понимать, что когда удаленная ветка ребейзена и вы делаете git pull --rebase базовый коммит — это не общий коммит в ветках feature и origin/feature, а первый незапушенный вами (то есть локальный) коммит. Линейка локальных коммитов, которая перемещается при git pull --rebase, легко конвертится в серию патчей, которые можно накатить, что бы с веткой не происходило — rebase, squash, такую серию патчей можно накатить куда угодно, поэтому я не понимаю, почему все так боятся ребейзить удаленные ветки.

В ситуации со вторичными ветками — нужно скорее брать за правило ребейзить и закрывать все вторичные бранчи, относительно первичных до их ребейза — и это будет работать просто прекрасно.

Не надо придумывать никаких универсальных административных политик. Цель статьи — чтобы после прочтения стало понятно, как физически работает rebase в описанных ситуациях. Владея этим знанием нужно применять их по обстоятельствам, а не по своду каких-то магических методик. Если у меня 1 локальное изменение и я взял git pull и сижу правлю огромное количество конфликтов в коде, который я даже не трогал — что-то тут пошло не так, да? Наверно нужно откатиться и взять git pull --rebase.

Нет никаких солюшенов и сводов правил, есть вы и git, некий набор инструментов в рамках гита и ваше умение ими пользоваться.

m00t, 7 декабря 2012 в 11:37 (комментарий был изменён) # ↑ 0

Конечно же, я имел ввиду git fetch && git rebase — опечатался там.

> Нет никаких солюшенов и сводов правил, есть вы и git, некий набор инструментов в рамках гита и ваше умение ими пользоваться.

Кажется, до меня дошла ваша мысль). Если есть понимание инструментов и умение ими пользоваться плюс база различных паттернов их использования, то для каждого проекта каждый сам сможем решить для себя, какие техники и когда применять.

Dr_Logic, 7 декабря 2012 в 22:22 # ↑ 0

Именно это я и хотел сказать — причем «паттерны» — это как бы частички вашего собственного опыта, а не вычитаные где-то рецепты. То есть их конечно можно вычитать, но понимать что и как именно они делают все равно надо, иначе если произойдет что-то непредвиденное разобраться самостоятельно будет крайне трудно. :)

alexanderzaytsev, 7 декабря 2012 в 11:47 # ↑ 0

Всегда делаю git fetch, а дальше по обстоятельствам. Я больше не доверяю я *git pull* и тем более *git pull --rebase*. Ну не нравиться мне потом в рефлоге ковыряться…

m00t, 7 декабря 2012 в 12:14 # ↑ 0

А расскажите подробнее, почему больше не доверяете? Я тоже всегда делаю fetch, а потом смотрю что дальше, но теперь хотел делать всегда git pull --rebase вместо ручного git fetch && git merge/rebase

alexanderzaytsev, 7 декабря 2012 в 12:44 # ↑ +1

Потому что ты не видишь что тебе пришло. Он просто берет и начинает ребейз, а потом говорит: упс конфликт. Т.е. его нужно прямо сейчас решить или откатиться.

При явном *fetch* я могу примерно оценить, будут конфликты или нет, бегло просмотрев сообщения и/или правки. Возможно при ребейзе нужно будет использовать какую-нибудь нестандартную стратегию. Нужны ли мне вообще эти изменения прямо сейчас, и т.д.

m00t, 7 декабря 2012 в 16:16 # ↑ 0

справедливо, спасибо

tsabir, 7 декабря 2012 в 11:04 # 0

Спасибо за статью!

Хотелось бы иметь некую выжимку из статьи, чтобы повесить на стенку. Этакая табличка в 2 столбика: (1) Разработчик, (2) Интегратор проекта, а в строках сверху вниз их действия в хронологическом порядке. Классный бы рефкард получился…

Dr_Logic, 7 декабря 2012 в 22:26 # ↑ 0

А вы распечатайте для начала эту и эту картинки и повесьте на стену. А лучше перерисуйте, а то у меня руки не очень прямые, да и цвета какие-то не очень выразительные вышли. Мне кажется такие вот «комиксы» куда наглядней, чем таблички.

garex, 7 декабря 2012 в 11:50 # +1

За коммиты в мастер/деволоп а-ля «Mergin remote into local» надо вообще на костре сжигать и точка.

ps: Ну ладно — но хотя бы поджаривать слегонца? :)

m00t, 7 декабря 2012 в 12:15 # ↑ 0

ДА!

solmasters, 7 декабря 2012 в 14:33 # 0

Спасибо за статью, познавательно. Только не deattached, а detached, исправьте, пожалуйста.

Dr_Logic, 7 декабря 2012 в 17:11 # ↑ 0

Спасибо, сейчас исправлю, писал по памяти.

cubuanic, 8 декабря 2012 в 05:31 (комментарий был изменён) # +2

cite: Если вы забудете указать идентификатор ветки, то force-push будет выполнен для всех локальных веток, имеющих удаленный оригинал.

Чтоб такого не случилось — в конфиге git'а параметру push.default надо поставить значение upstream или current (это разные вещи, выберите сами что вам больше подходит).

git config --global push.default upstream
git config --global push.default current

Значение по-умолчанию — matching, и я категорически не понимаю, почему это именно так.

Dr_Logic, 8 декабря 2012 в 13:17 # ↑ 0

Спасибо! Отличное замечание. Чуть позже добавлю в текст самой статьи.

edelweard, 11 декабря 2012 в 19:03 # ↑ 0

Между прочим, в Git 2.0 поведение по-умолчанию будет наконец-то изменено с matching на current.
Сообщение (warning) по этому поводу выдаётся, если у вас не прописан push.default, начиная с Git 1.8.

Xenkok, 11 марта 2013 в 18:46 # 0

Можно задать настройки для автоматического rebase.

Использовать автоматический rebase для новых веток:
```
git config branch.autosetuprebase always
```

для существующих:
```
git config branch.*branch-name*.rebase true
```

AFoST, 20 мая 2014 в 18:31 (комментарий был изменён) # **0**

[del]

Только зарегистрированные пользователи могут оставлять комментарии. Войдите, пожалуйста.