Безболезненное разрешение Merge конфликтов в Git

Предлагаю читателям "Хабрахабра" перевод публикации ["Painless Merge Conflict Resolution in Git"](http://blog.wuwon.id.au/2010/09/painless-merge-conflict-resolution-in.html)из блога blog.wuwon.id.au.

В моей повседневной работе, часто приходится иметь дело со множеством git ветвей (branch). Это могут быть ветви промежуточных релизов, ветви с устаревшим API находящиеся на поддержке для некоторых клиентов, или ветви с экспериментальными свойствами. Лёгкость создания ветвей в модели Git так и соблазняет разработчиков создавать все больше и больше ветвей, и как правило бремя от большого количества ветвей становится очень ощутимым, когда приходится все эти ветви поддерживать и периодически делать слияния (merge) с другими ветвями.

Слияния очень важны для поддержания кода в актуальном состоянии, и как правило ошибка сделанная при слиянии может привести к большей головной боли, нежели ошибка сделанная при простом коммите. К сожалению ошибки слияния далеко не редкость, потому что во-первых слияния имеют несколько родительских ветвей. Даже при анализе истории слияния ветвей, бывает очень трудно понять, какие же изменения были сделаны для разрешения конфликта. Во-вторых, [отмена неудачного слияния](http://www.kernel.org/pub/software/scm/git/docs/howto/revert-a-faulty-merge.txt) может превратиться в большую головную боль. В-третьих, большая часть конфликтов слияния происходит при работе с чужим кодом, потому что само понятие ветвей подразумевает множество пользователей, т.е. далеко не всегда слияние производит тот же человек который работал с той или иной веткой. В сухом остатке, сделать ошибку при слиянии очень легко, её трудно исправить и трудно найти. Таким образом время потраченное на изучение и понимание процесса слияния ветвей, окупится с лихвой.

Удивительно, но я обнаружил, что многие доступные инструменты и интерфейсы предназначенные для выполнения слияний, не достаточно хорошо оснащены для эффективного выполнения этого процесса. Часто программист просто надеется что команда git merge сделает за него всю работу. Но когда все-таки происходит конфликт, то обычно стратегия слияния заключается в беглом просмотре кода вокруг строки конфликта, и интуитивном угадывании что именно данный кусок кода предпочтительней другого.

В данной статье я надеюсь продемонстрировать что процесс разрешения конфликтов может быть пошагово точным, при котором отпадает необходимость что-либо там угадывать.

## **Голубые Розы (Roses are Blue)**

Давайте предположим что вашей команде поручили писать поэмы в отведённом для этих целей репозитории. (Какой кошмар!) А вам доверили самое главное — делать слияния последних фиксов из ветки master в ветку beta. Итак, вы переключаетесь в ветку beta и выполняете следующую команду:

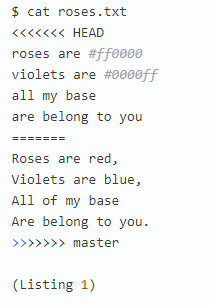
$ git merge master

Auto-merging roses.txt

CONFLICT (content): Merge conflict in roses.txt

Automatic merge failed; fix conflicts and then commit the result.

Ого, это конфликт. Вы решаете просмотреть файл на который ссылается git:



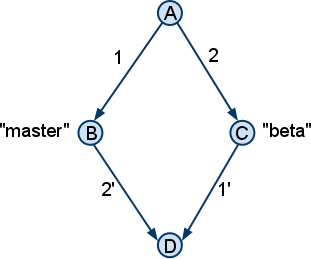
Замечательно! Весь файл, как показывает Listing 1, находится в конфликтном состоянии. Какой же вариант файла является более корректным? Оба варианта выглядят корректно. Верхний вариант написан в хакер-стиле с элементами цветовой кодировки в стиле HTML и с использованием только строчных букв. Нижний вариант выглядит более натурально, с использованием пунктуации и заглавных букв.

Если бы это был ваш проект, вы бы могли просто выбрать один вариант и покончить с этим слиянием. Но проблема в том, что это не ваша поэма, вы никогда не читали эту поэму раньше, не были ответственны за написание или редактирование, и вы отлично понимаете что в случае не верного решения чья-то тяжёлая работа может кануть в небытие. Однако вас всё же назначили ответственным по слиянию этих веток. Что же вам делать?

## **Назад к Базе (Back to Base)**

Хитрость заключается в том, что Listing 1 не даёт вам полную информацию, необходимую для совершения корректного слияния. На самом деле, в процессе слияния участвуют четыре важных части информации (состояния), три из которых просто необходимы для успешного разрешения конфликта. В случае Listing 1, Git предоставил вам только два состояния.

Следующая диаграмма иллюстрирует эти четыре состояния:



Состояния (B) и © относятся к текущим положениям (head) веток master и beta соответственно, эти два состояния как раз таки и отражены в Listing 1. Состояние (D) это результат слияния, то что вы хотите получить/сгенерировать в конечном итоге (в большинстве случаев Git автоматически генерирует состояние (D)). Состояние (А) на самом верху, представляет собой базу (основу) слияния веток master и beta. База слияния (A) это последний общий предок веток master и beta, и пока предположим что это база слияния уникальна. Как мы увидим позже состояние (A) играет ключевую роль в разрешении конфликтов. На диаграмме я также отразил дельты 1 и 2, которые представляют изменения между состояниями (A)-(B), и (A)-© соответственно. Зная состояния (A), (B) и © дельты 1 и 2 могут быть легко получены (вычислены). Обратите внимание, что дельты 1 и 2 могут состоять из более чем одного коммита. Но для наших целей будем считать что все дельты монолитны.

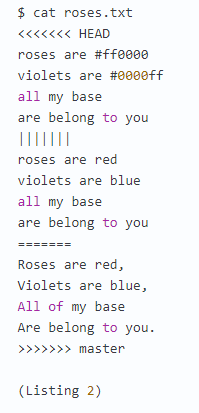
Чтобы понять, как получить состояние (D), вы должны понимать что же операция слияния пытается сделать. Состояние (D) должно представлять собой сочетание изменений, внесённых в ветку master и beta соответственно. Т.е. другими словами сочетание дельт 1 и 2. Идея проста на поверхности и большую часть времени не требует вмешательства со стороны человека, за исключением особых случаев когда дельты затрагивают наслаиваемые (пересекающиеся) части файла. В такой ситуации вам требуется помочь машине сгенерировать результат (D), путём сравнения дельт 1 и 2.

## **Определение Отличий (Identifying the Differences)**

Для того чтобы найти изменения внесённые в каждую ветку, необходимо знать как выглядит база слияния, состояние (A). Самый простой механизм получения информации о базе слияния, это установка опции merge.conflictstyle в значение diff3

$ git config merge.conflictstyle diff3

После включения этой опции, попробуйте заново сделать слияние (git reset --hard; git merge master) и проинспектируйте конфликтующий файл ещё раз:



Теперь мы видим третий фрагмент посередине, который и является базой слияния или состояние (A). Изменения видны как на ладони: в ветке beta (HEAD) человеческие названия цветов были заменены на HTML коды, а в ветку master добавили капитализацию и пунктуацию. Основываясь на этих знаниях, мы теперь знаем что результат должен включать в себя капитализацию, пунктуацию и HTML коды цветов.

В принципе на этом можно было бы и закончить, потому что результат достигнут. Но есть решение и получше.

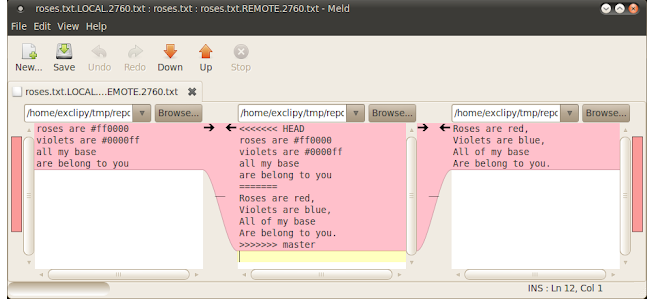
## **Графическое Слияние (GUI Merging)**

Хотя и простое текстовое представление конфликта слияния делает свою работу в простых случаях, на практике конфликты могут быть более радикальными и сложными. В таких случаях могут помочь графические инструменты. Мой выбор пал на простой инструмент написанный на Python под названием [meld](http://meld.sourceforge.net/), но может подойти любой другой графический инструмент, способный представить слияние в трёх-колоночном виде.

Для использования графического инструмента (он должен быть установлен), после того как git пожаловался что есть конфликт, введите следующую команду:

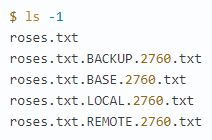
$ git mergetool

Последует вопрос какой программой для слияния вы хотели бы воспользоваться, просто введите meld и нажмите Enter. Вот как окно программы может выглядеть (подразумевается опция merge.conflictstyle не была включена):



Несмотря на то что информация представлена бок о бок, она не отображает нужные фрагменты которые были в Listing 2. Мы не видим здесь фрагмента базы слияния (состояния (A)), что мы видим это файл roses.txt.LOCAL.2760.txt в левой колонке и файл roses.txt.REMOTE.2760.txt в правой колонке и файл посередине это неудачное слияние. Т.е. по сути нам представили состояния (B), © и несостоявшееся состояние (D), но состояние (A) отсутствует...

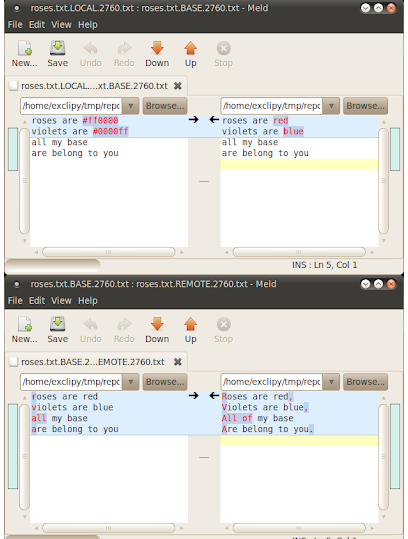
Правда отсутствует? Давайте проверим, в старом добром терминале:



Видим интересующий нас файл: roses.txt.BASE.2760.txt. Это и есть файл базы слияния. Теперь нам осталось всего лишь найти изменения внесённые в ветки master и beta, по отношению к базе. Мы можем сделать это двумя отдельными вызовами meld:

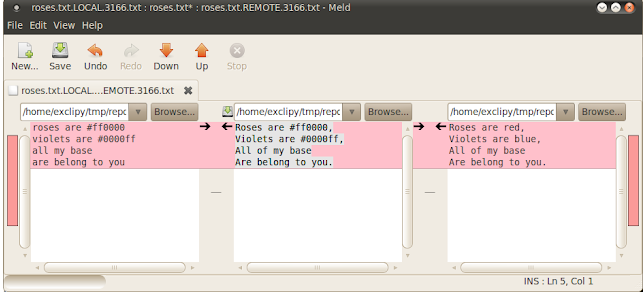


(Кто-то может подметить что было бы более разумно, поменять порядок аргументов в первом вызове, для того чтобы файл базы находился в левой колонке в обоих случаях, но именно такой порядок сохраняет подобие трёх-колоночного вида, при котором база остаётся по середине.) Результат выполнения — два окна как показано ниже:



При чтении первого окна справа налево и второго окна слева направо, становится ясно как день, какие изменения произошли в каждой ветке. Так как meld любезно подсветил все изменения, теперь практически не возможно пропустить даже мелко заметные правки (Кто-нибудь заметил добавление предлога "of" при просмотре текстового представления разрешения конфликта Listing 1 или даже Listing 2?)

Вооружившись этими знаниями, мы теперь можем вернуться к трёх-колоночному представлению и сделать изменения. Моя стратегия ручного слияния это взять весь текст из ветки с более весомыми изменениями (в данном случае master/REMOTE т.е. beta), и поверх него производить пошаговые правки, т.е. вносить изменения сделанные в другой ветке (master). Вот что получилось:



## **А теперь всё вместе (All Together Now)**

Надеюсь, вы найдёте этот трёх-окошечный метод разрешения конфликтов, таким же полезным каким нахожу его я. Но согласитесь что запускать новые вызовы meld вручную каждый раз при разрешении конфликтов, не очень то и удобно. Выход, это настроить git таким образом чтобы все три окна открывались автоматически при вызове команды git mergetool. Для этого можно создать выполняемый скрипт, который должен находится в переменной окружения PATH (например $HOME/bin/gitmerge), со следующим содержимым:



После того как вы привыкните к такому разрешению конфликтов с использованием трёх вышеупомянутых окон, вы скорее всего обнаружите, что процесс стал более методичным и механическим. В большинстве случаев, вам даже не придётся читать и понимать куски кода из каждой ветки, для того чтобы понять какой же вариант применить для слияния. Вам больше не понадобится догадываться, потому что вы будете гораздо более уверенным в корректности вашего комита. Из-за этой уверенности, появится чувство что разрешение конфликтов превратилось в увлекательное занятие.

### **Бонус от переводчика**

Для тех кто пользуется tmux и n?vim, предлагаю следующий скрипт [gitmerge](https://github.com/vbauerster/dotfiles/blob/master/bin/gitmerge):

#!/bin/sh

sn=gitmerge

tmux new-session -d -s "$sn" -n "diff3" "nvim -d $2 $4 $3"

tmux split-window -t "$sn:1" -v "nvim -d $2 $1"

tmux split-window -t "$sn:1" -h "nvim -d $1 $3"

**Примечание**: если вы не используете [эту опцию](https://github.com/vbauerster/dotfiles/blob/100cd6e3045a8d78a6a82f0b88dd51da1d827713/tmux.conf#L51-L52) в своем ~/.tmux.conf, то вам надо поменять в двух последних строках "$sn:1" на "$sn:0"

Соответственно добавьте следующее в ваш [~/.gitconfig](https://github.com/vbauerster/dotfiles/blob/100cd6e3045a8d78a6a82f0b88dd51da1d827713/gitconfig#L41-L44)

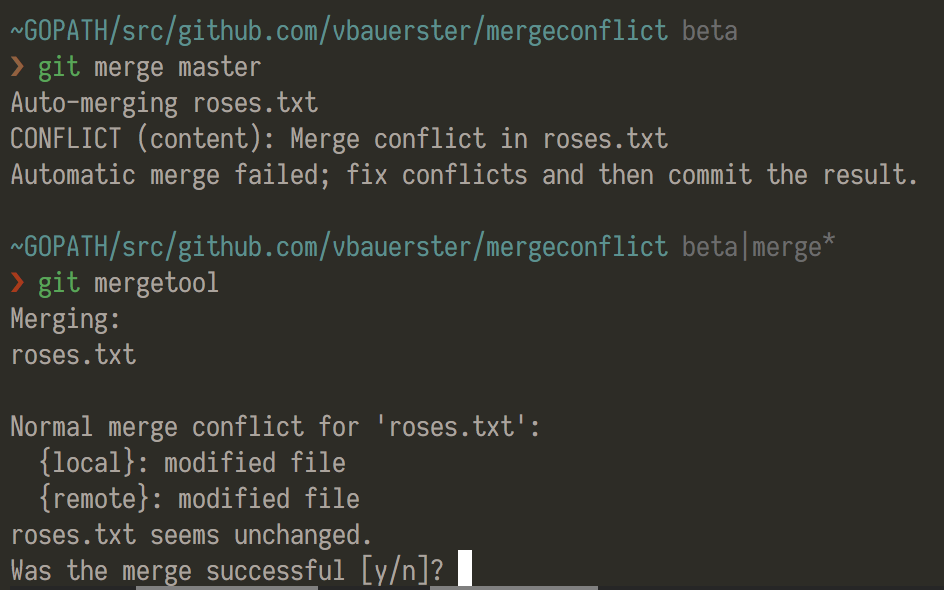
[mergetool "gitmerge"]

cmd = $HOME/bin/gitmerge \"$BASE\" \"$LOCAL\" \"$REMOTE\" \"$MERGED\"

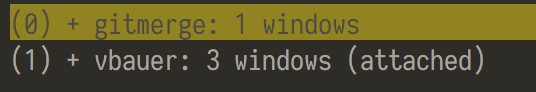
[merge]

tool = gitmerge

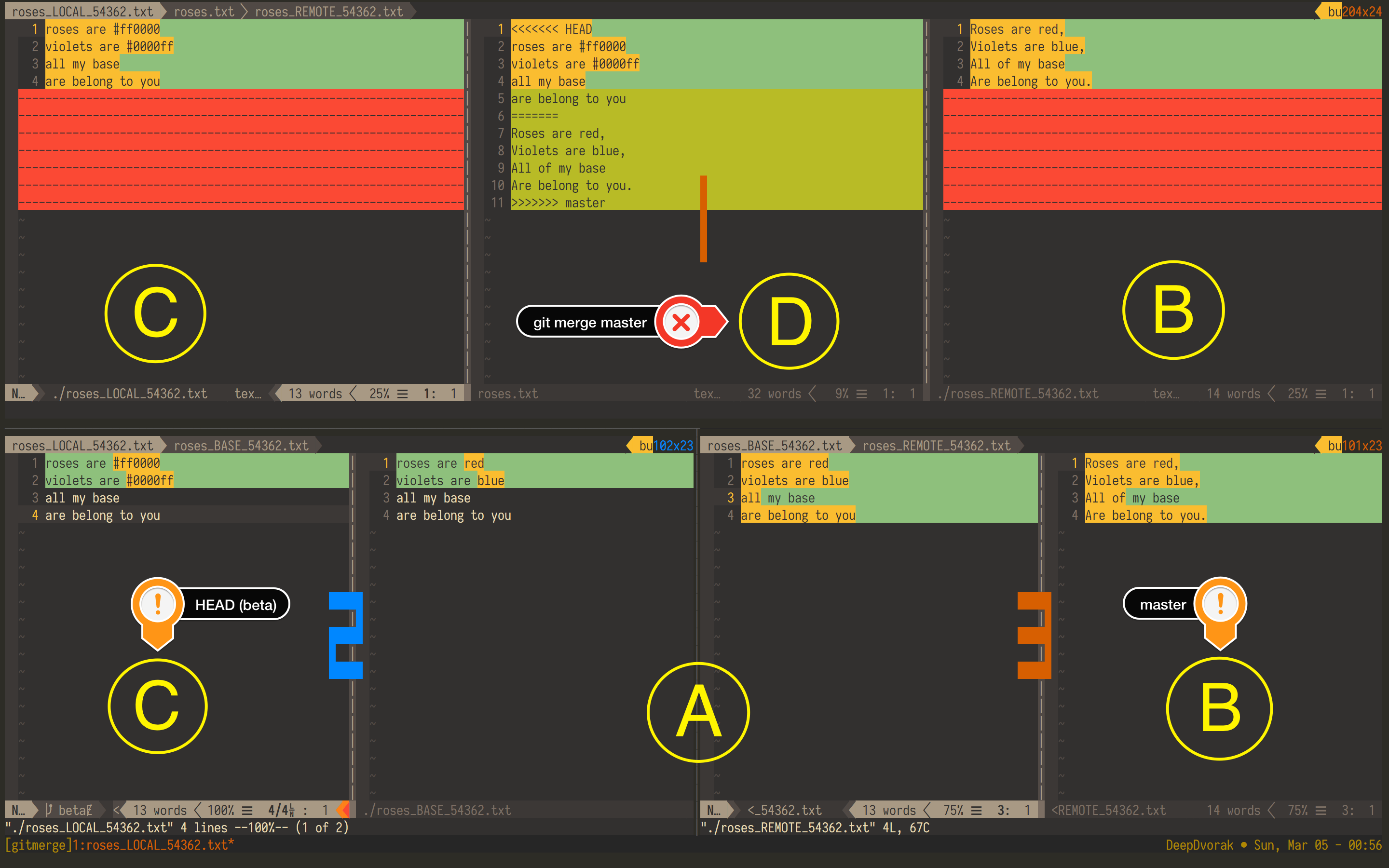
Воркфлоу разрешения конфликта будет выглядеть так:



Пока игнорируем вопрос (Was the merge successful [y/n]?) и переключаемся в сессию под названием gitmerge (сочетание TMUXPREFIX + s):

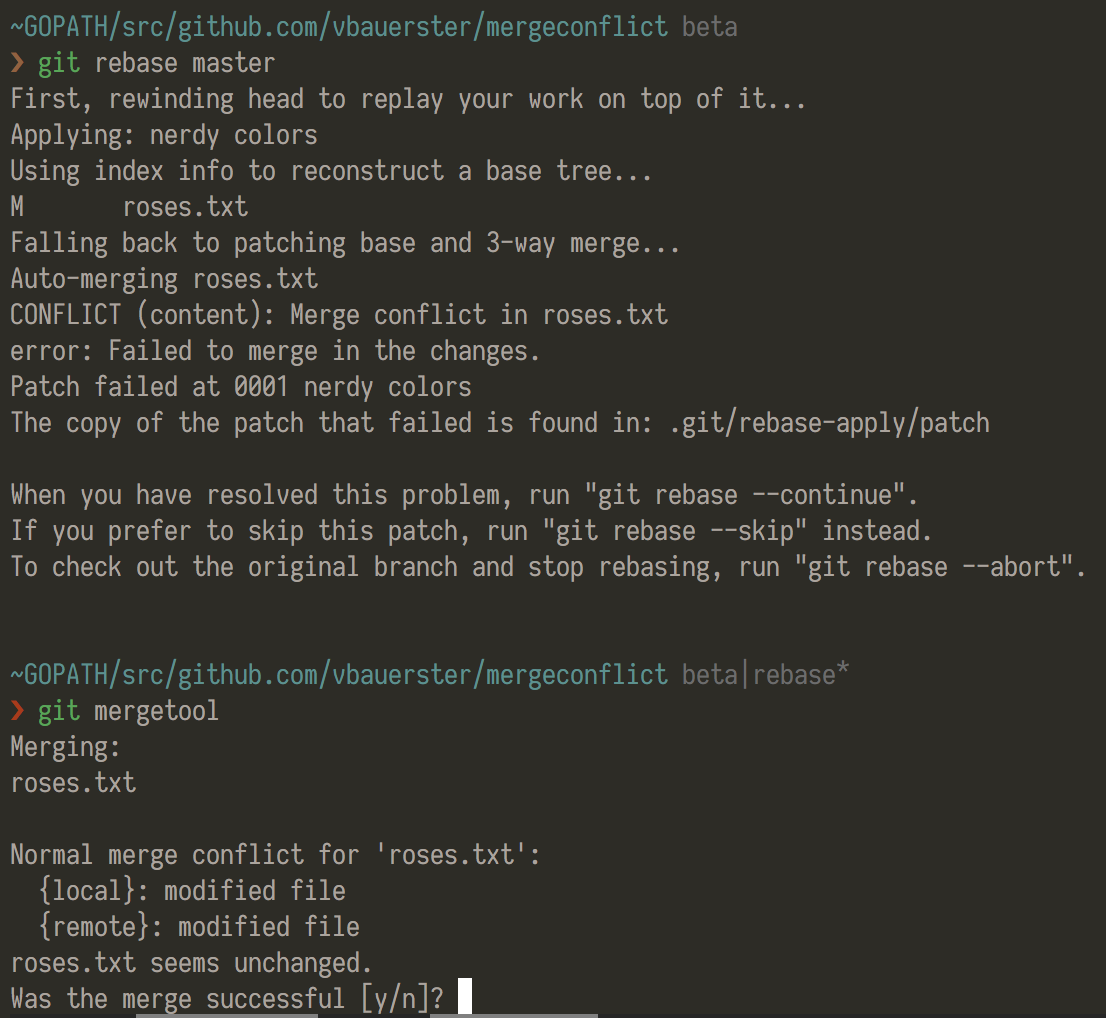


Видим наше трёх-оконное представление на одном экране. Цифрами обозначены сплиты (panes) tmux'a, буквами соответствующие состояния. Делаем правки для разрешения конфликта, т.е. редактируем состояние (D) и сохраняем. После этого возвращаемся обратно в исходную сессию tmux'a и подтверждаем что слияние произошло успешно.

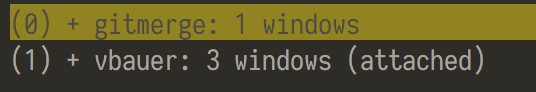


### **git rebase master**

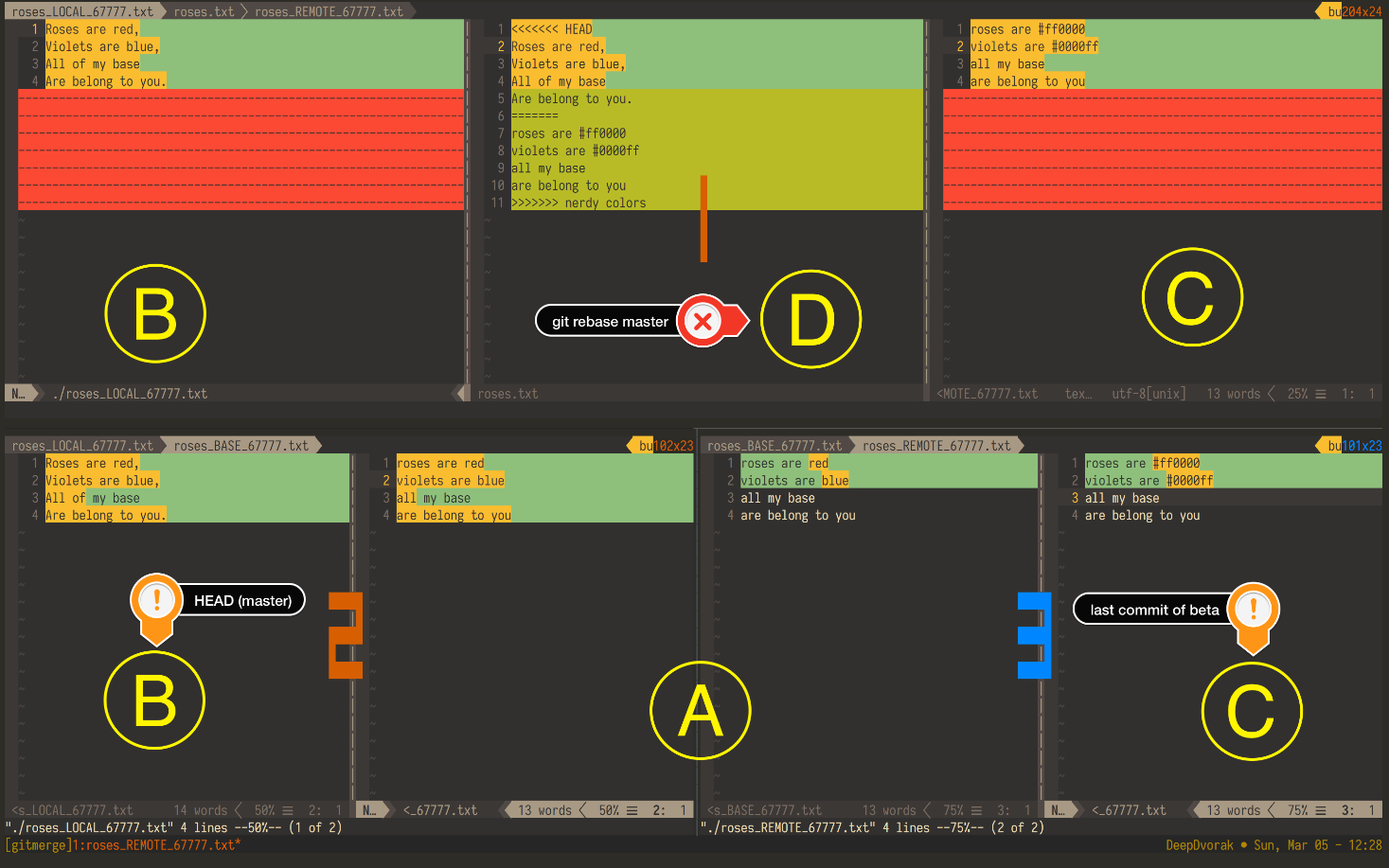
Лично я предпочитаю и считаю более правильным делать сначала rebase master в ветке beta, и только после этого переключаться в master и делать git merge beta. В принципе воркфлоу не сильно отличается, за исключением трёх-оконного вида.



Переключаемся в сессию gitmerge



Обратите внимание, что состояния (B) и © поменялись местами:



Рекомендую всем поиграться с [примером репозитария](https://github.com/vbauerster/PainlessMergeConflict/blob/master/mergeconflict.tgz) хотя бы один раз, сделать разрешение конфликта по вышеописанной схеме. Лично я больше не гадаю а что же выбрать "Accept theirs" или "Accept yours".