Система контроля версий (VCS) – система хранения, которая фиксирует изменения в виде версий, хранит историю изменений и управляет версионированием.

**Виды VCS**:

*Централизованные* – все завязано на сервере (полная история изменений находится только на сервере).

*Распределенные* – каждый клиент имеет всю историю изменений и все операции выполняются локально, серверная часть необязательна, но ей пользуются для удобства синхронизации.

Git – распределенная система.

***Области git:***

|  |  |
| --- | --- |
| Рабочая директория | Файлы и папки подконтрольные git |
| Staging area (индекс) | Промежуточная зона между рабочей зоной и непосредственно коммитом. Необходима для дополнительной проверки при коммите и позволяет разделять коммиты на более мелкие, для их группировки или написания отдельного комментария для каждого коммита. |
| Репозиторий | Место хранения всей истории разработки проекта |

\*Рабочая директория – папка, содержащая папку. git, индекс и репозиторий – находятся в папке. git

\*В индекс нельзя поместить пустые директории, для обхода создается пустой файл с именем .gitkeep

***Статусы -*** отражают состояние файлов в областях Git.

2 категории файлов в рабочей директории:

1) Tracked – файлы под версионным контролем   
2) Untracked – файлы, которые не нужно отслеживать, такие файлы должны быть помещены в файл .gitignore в виде шаблонов или имен.

Состояния Tracked файлов:

|  |  |
| --- | --- |
| Untracked | Новый файл, про который в репозитории еще не записей |
| Modified | Файл уже находится под контролем Git, но его состояние изменилось |
| Staged | Файлы, подготовленные для коммита, перемещенные в зону индекс |
| Deleted | Файл был удален из рабочей директории, но еще находится в репозитории |
| Unmodified | Файлы, состояние которых в рабочей директории и репозитории равны |
| Commited | Текущий снимок файла в рабочей директории соответствует репозиторию. Не является статусом, а скорее состоянием файла |

**Объекты git:**

|  |  |
| --- | --- |
| Blob | Бинарное представление файла, не содержит метаданных (не привязан к именам и путям). |
| Tree | Представление каталога, содержит список записей, соответствующий иерархии файловой системы, т.е ссылки на блобы и ссылки на поддеревья, а также имена этих структур и права доступа. |
| Commit | Сохраненное состояние репозитория в определенный момент времени (снимок).  Технически является указателем на корневое дерево (Включает ссылки на деревья и блобы, отражающие текущее состояние проекта).  **Данные в коммите**: метаданные автора и коммитера, описание коммита, ссылка на родительские коммиты (все коммиты связаны между собой посредством этого механизма, как односвязный список), ссылка на корневое дерево. |
| Annotated Tag | Метка обозначающая важные события в репозитории |

**Хэширование:** Длякаждого объекта Git генерируется SHA-1 хэш – 40 символов в 16-ричной системе (20 байт), такой размер хэш уменьшает вероятность коллизий. Хэш = ссылка на объект Git.

Генерируется на основании содержимого объекта, то есть для Blob – его содержимое, для Tree – хэши его содержимого и метаданные этого содержимого (права доступа и имена), для commit – соответственно, его содержимое (метаданные, корневое дерево, родительские коммиты). При изменении любого компонента коммита изменится хэш этого компонента, что приведет к изменению хэш коммита.

**Механика:**

Коммит сопровождается созданием **корневого дерева** – представление всей структуры объектов (точка входа в содержимое снимка), далее создаются новые объекты и рекурсивно определяются объекты хэш которых был изменен и также создаются объекты для них. Объекты, которые не были изменены переиспользуются путем получения ссылок на них из предыдущих коммитов. Изменения в объектах определяются путем сравнения их хэшей. Далее коммит связывается ссылкой со своими родительскими коммитами.

**Указатели:**

1. Head – указатель на коммит (чаще на ветку), который будет отражать состояние рабочей директории, создание нового коммита произойдет от этого указателя.
2. Указатель на ветку – указатель на последний коммит в ветке, перемещается вместе с head, если он находится в текущей ветке (A->B->C , где C – последний коммит в ветке, checkout head на B и создание коммита A->B->D приведет к перемещению указателя ветки на новый коммит D, а коммит C больше не будет принадлежать этой ветке).
3. Ветка слежения - указатель, необходимый для синхронизации локального состояния репозитория и удаленного.
4. Тэги – статичные указатели на важные коммиты – аннотированные (полноценный объект гит с параметрами) и легковесные (просто указатель без метаданных).

Достижимость коммита – коммит считается достижимым, если принадлежит хотя бы одной ветке, если он не включен ни в одну из существующих веток, такие коммиты с течением времени (30 дней) будут удалены GC.

**Командная строка**

|  |  |
| --- | --- |
| pwd | Абсолютный путь |
| Cd без аргументов  Cd с аргументами  ..  ../.. | в домашний каталог  в указанный  на одну директорию назад  на 2 назад и так далее |
| mkdir | Создает директорию |
| Echo “какой-то текст”> имя файла.расширение | Создать файл |
| ls | Содержание текущей папки |
| cat | Прочитать файл |
| rm | Удалить файл |
| Ls -la | Флаг - la позволяет увидеть скрытые папки |
| vim | Вызов редактора |
| touch | Создать пустой файл |

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Описание*** | ***Флаги*** | ***Описание*** |
| **Инициализация Git Репозитория** | | | |
| init | Создание Git репозитория | | |
| ***Команды для настроек и отображения информации*** | | | |
| config | Установка параметров и конфигураций.  Задается уровень видимости конфигурации:  system – все пользователи системы; | --list | список всех настроек Git |
| global – конкретный пользователь;  local – текущий репозиторий;  Пример:  git config –global set.name “some name” | alias | задать написание для команды собственным словом/символом.  Пример: git config –global alias.c ‘config –global’  Теперь команда config --global вызывается с помощью <c> |
| show | Детально отображает указанный коммит – все его изменения | <hash-commit> | Отображает инфу по указанному коммиту, если не указывать коммит, то выведет HEAD. Hash-commit можно указывать первые 4 символа, а не весь. |
| <hash-commit >:<путь к файлу> | Выводит не отличия, а полностью файл в текущем коммите |
| status | Отражает текущее состояние Git репозитория |  |  |
| log | Отображение истории коммитов Отображает историю коммитов по отношению к HEAD (подразумевается по умолчанию, может быть любая ветка или коммит) | -p | Работает как show/diff, но для всех коммитов (отображает подробности изменений для каждого коммита) |
| -p -2 | Выведет 2 последних коммита |
| --oneline | Покажет коммиты без метаинформации (только Hash и комментарий) |
| --graph | Отображает графически логи |
| --all | Выводит логи, достижимые из всех веток |
| <Ветка1>..<Ветка2> | Отобразит коммиты Ветка2, которых нет в Ветка1 – то же, что и <Ветка2>^<Ветка1> |
| <Ветка1>...<Ветка2> | Отображает коммиты веток, кроме общих (флаг --boundary отобразит пограничный коммит, -- graph при использовании … сработает только с ним git log <Ветка1>…<Ветка2> --boundary --graph) |
| <Имя> | Выведет коммиты, в которых менялся файл |
| --grep<слово> | Поиск коммитов по слову в описании (использует регулярные выражения) |
| -G | Ищет совпадения в файлах |
| reflog | Команда, для обращения к файлу с записями о перемещениях указателей веток и HEAD. Рефлоги хранятся 90 дней (30 для недостижимых коммитов), хранятся локально и не отправляются в GitHub.  Является аналогом git log -g |  | С помощью данных, имеющихся в reflog, можно переключаться на коммиты не прибегая к указанию на хэш, возможно использование идентификатора, который хранится в рефлогах вида HEAD@{1} или дата вида HEAD@{2017-09-12}) |
| diff | Показывает все изменения содержания файлов (игнорирует неотслеживаемые файлы) | HEAD | Сравнивает содержимое рабочей директории и текущим коммитом |
| <Без флага> | Показывает разницу между Рабочей директорией и индексом |
| –staged/cached | показывает изменения между индексом и репозиторием (обычно делают для просмотра того, что планируется закомитить) |
| <Коммит><Коммит> | Различия между двумя коммитами |
| <Коммит><Коммит> <Имя> | Различия между двумя коммитами для этого файла |
| --word-diff | Покажет не строки, которые были изменены, а конкретно измененные слова |
| blame | Показывает автора каждого изменения для файла |  |  |
| **Команды для модификации областей** | | | |
| add | Помещает указанный файл в индекс | < .>-пробел точка | Поместит все модифицированные и неотслеживаемые файлы в индекс |
| -f | Добавляет в индекс, даже если файл в .gitignore |
| -p | Режим выбора изменений для добавления их в индекс. В файле 2 изменения в рабочей директории, можно выбрать, какое добавлять в индекс, а какое нет |
| commit | Переносит снимок изменений в репозиторий (можно указывать имена файлов)  Коммит должен быть атомарным (один коммит – одна цель) и консистентным (полностью завершенным). | Без флага | Откроется редактор для написания комментария к коммиту – Первая строка должна быть заголовком (не более 50 символов без точки) далее пустая строка, последующие строки начинаются с \* и комментируют детали изменений |
| -m <Сообщение коммита> | Оставить короткий коммент на коммит |
| -v | Вызовет diff --staged вместе с сообщением для коммита |
| -a/-all | Коммитит ВСЕ файлы, сразу из рабочей директории (работает только для модифицированных файлов) |
| --amend | Быстрая замена текущего коммита.  Содержит в себе эффект 2х команд: git reset --soft @~ и git commit  То есть сначала отступает на 1 шаг назад, а затем создает новый коммит, применяя текущие изменения в индексе. |
| --author | Установить автора для коммита – метаданные коммита включают автора файла и коммитера |
| --fixup=<коммит> | Создает коммит-заплатку, с отредактированным заголовком (добавляет «fixup!») |
| - c <Хэш коммита>  -С <Хэш коммита> | Подтянет описание (коммент и коммитера) из указанного коммита (откроет редактор)  Тоже самое, но без редактирования |
| clean | Для удаления неотслеживаемых файлов и директорий.  Необходима, т.к. команды типа checkout или reset работают только с файлами и папками отслеживаемыми Git. | -f | Без него не работает |
| -n | Список файлов к удалению |
| -d | Удаление директорий |
| -x | Удаление файлов, включенных в gitignore |
| mv | Перемещение файла по директориям или переименование | Git mv read.txt another.txt  Git mv read.txt путь к директории – если указать файл, то этот файл еще и переименуется | Если указать имя, то переименование  Если путь – перемещение  Можно комбинировать эти операции |
| stash | Временное сохранение незакомиченных изменений (изменения архивируются и исключаются из рабочей директории и индекса), тем самым позволяет переключаться между ветками при наличии незакомиченных изменений (сохраненные изменения не привязаны к веткам, то есть наложить их можно куда угодно). | Save  List  Apply  Pop  Drop  clear |  |
| restore | Возвращает рабочую директорию к состоянию последнего коммита | <Имя файла>  < .>  -- staged | Для файла  Для всех файлов  Очищает только индекс |
| rm | Удаляет файл или папку из рабочей директории и добавляет изменения в индекс. | -f или --force | Для удаления файлов, изменения которых не были сохранены в репозитории.  Если файл был изменен перед удалением, то git не даст его удалить без этого флага, т.к. изменения потеряются |
| --cached | Удаляет из индекса, но не удаляет файл из рабочей директории. |
| -r или --recursive | Рекурсивно удаляет папку (сначала содержимое, потом папку) |
| **Команды для перемещения указателей, управления коммитами** | | | |
| branch | <Без параметров> Показывает все локальные ветки (не показывает ветки удаленного репозитория) | <Имя ветки> | Создает новую ветку от текущей позиции HEAD |
| -a | Показывает локальные и удаленные ветки |
| -v | Тоже самое, только с хэш-суммой |
| -d | Удаляет ветку (не сработает, если в ней есть коммиты, которые в случае удаления текущей ветки не будут принадлежать ни к одной ветке – недостижимые коммиты) |
| -D | Удаляет ветку даже в случае наличия коммитов не принадлежащих к другим веткам |
| -f <Имя><Коммит> | Принудительное пересоздание ветки на указанном коммите в случае существования ветки с таким именем (используется для перемещения указателя ветки на другой коммит, при этом нужно находиться на другой ветке) |
| checkout | Перемещает указатель HEAD и приводит рабочую директорию к состоянию этого коммита. Можно использовать как на ветке, так и на отдельном коммите (состояние отделенный Head).  Команда не позволяет переключиться на другую ветку, если в текущей есть незакомиченные изменения. Исключение, если файл, в котором произошли изменения не был изменен в коммите, на который переключались  Состояние отделенный Head – возникает в случае, если переключаться не на ветку, а на коммит, в таком случае коммиты от этой точки не будут принадлежать ни одной из веток и впоследствии подлежат удалению. | -b <Имя> | Создает ветку <Имя> и перемещает на нее HEAD (не нужно вызывать сначала branch а потом checkout) |
| -b <Хэш коммита>  -B <Хэш коммита> | Создаст ветку от указанного коммита (не сработает, если ветка существует)  Сработает с существующей веткой (для перемещения указателя ветки) |
| <Имя файла> | Сбрасывает состояние файла до состояния последнего коммита. |
| -f | Перемещает указатель HEAD и затирает все изменения в репозитории и индексе (переключение командой checkout –f HEAD останется на том же коммите, но удалит все изменения из рабочего и индекса) |
| <Хэш коммита><Имя файла> | Подтягивает версию файла из указанного коммита в рабочую директорию и индекс |
| tag | Команда для создания и работы с тэгами.  git tag <без аргументов> - список тэгов.  Тэги не пушатся вместе с коммитами на сервер автоматически. | <Имя> <Хэш коммита> | Создание легковесного тэга, который указывает на коммит. |
| -n | Добавляет к выводу сообщения, принадлежащие соответствующему коммиту |
| -d | Удаление тэга |
| –a <имя> -m”сообщение” | Аннотированный тэг |
| push <origin> <имя тэга> | Запушить указанный тэг |
| push –tags | Запушить все тэги |
| reset | Работает только с отслеживаемыми файлами  [\*Описание работы и всех флагов\*](#reset) | --hard <Хэш коммита>  --soft <Хэш коммита>  --mixed <Хэш коммита> |  |
| cherry-pick | Копирование коммита и его перенос на текущий HEAD.  При применении к нескольким коммитам с помощью явного указания коммитов или диапазона типа master..feature, коммиты будут последовательно применяться к текущему Head | -x | Добавляет в сообщения коммита строку о том, откуда был взят коммит (его хэш) |
| -n / -no-commit | Так же как и у [merge](#Семантика) |
| Команда, так же как и merge, может закончится либо созданием нового коммита, либо конфликтом, который нужно разрешить вручную | |
| --abort | Откат всех изменений на состояние до применения cherry-pick |
| --quit | Изменения, которые прошли без конфликта сохраняются, конфликтные откатываются |
| --continue | Продолжение процесса перемещения |
| merge | [\*Слияние веток\*](#Merge) Указатель Head на той ветке в которую сливаем изменения, <Имя ветки> из которой сливаем |  |  |
| rebase | [\*Перебазирование\*](#rebase) |  |  |
| revert | Используется для отмены изменений в одном или нескольких коммитах путем создания нового коммита с противоположными им изменениями (коммит отмены). revert <Хэш коммита>  Если нельзя изменять историю и она важна для других участников | -m <Родитель> | Используется для указания родительской ветки, которая будет являться основной при revert для merge коммита (т.к. изменения подтягиваются из 2х веток). К изменениям других веток будет применен коммит отмены. Родители указываются цифрами (1 – основной родитель, в которого вмержили ветку) |
| **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | | | |
| ^ | Знак исключения |  |  |
| :/ | поиск |  |  |
| Gui | Вызов графического интерфейса Gui | gui& | Вызовется в новом окне |
| ORIG\_HEAD | Содержит предыдущую ссылку на HEAD |  |  |
| @ | Обозначение указателя HEAD |  |  |
| ~ | Назад на 1 | ~3 | 3 коммита назад |
| **Команды для Git Hub** | | | |
| clone | Клонирует репозиторий по ссылке и выполняет remote |  |  |
| remote | Управляет удаленными репозиториями | -v | Показывает список всех связанных с компьютером репозиториев |
| add <имя> <Ссылка на репозиторий> | Связывает новый удаленный репозиторий. Имя – задаем имя этой ссылке, обычно origin |
| remove<Имя> | Удаление удаленного репозитория |
| rename <Старое имя><Новое имя> | Переименование удаленного репозитория |
| push | Отправляет изменения на связанный удаленный сервер | <имя удаленного сервера><ветка>  -u –связывает ветку в локальном и удаленном репозиториях, теперь можно вызывать pull и push без пути | При выполнении команды push, выполняется сравнение веток удаленного и локального репозиториев посредством положения указателя ветки слежения, если они не равны, то требуется выполнить pull (под капотом вызывается fetch и merge)  указатель нельзя перемещать вручную, двигается только при команде pull/push |
| pull | Подтягивает изменения с сервера на комп | <имя удаленного сервера><ветка> |
| fetch | Обновления состояния веток слежения (подгрузит недостающие коммиты в случае несовпадения) |  | При подгрузке проекта он отображает только ветку мастер, чтобы подгрузить другую ветку надо переключиться на нее, то есть, если подгрузить проект, с несколькими ветками, потом вызвать branch, видно будет только ветку master |
| Pull request/merge request – запрос на слияние веток, то есть настройки (branch protection) не позволяют просто так вмержить в эту ветку другую без предварительного одобрения. (fork + pull, модель общего репозитория) | | | |

***Команда reset***

В общем смысле команда перемещает указатель ветки и указатель HEAD на указанный коммит.

3 основных флага:

--hard перемещает указатель ветки на указанный коммит и приводит рабочую директорию и индекс к состоянию репозитория этого коммита (полностью очищает рабочую директорию и индекс).

Если коммит не указан, то подразумевается HEAD, тогда флаг перемещен не будет, а просто затрутся изменения в рабочей директории и индексе Git reset –hard (аналог git restore . + git restore --staged)

--soft перемещает указатель ветки на указанный коммит, при этом совершает «обратный коммит», то есть разница между текущим коммитом и коммитом, на который происходит перемещение, будет помещена в индекс, при этом все незакомиченные изменения также переносятся. В случае конфликта незакомиченных изменений и изменений из текущего коммита приоритетными будут незакомиченные изменения.

--mixed – режим reset по умолчанию, перемещает указатель ветки на указанный коммит, полностью сбрасывает состояние индекса (переносит изменения в рабочую директорию), а рабочая директория остается неизменной (включая незакомиченные изменения). Применяется для очистки индекса текущего коммита (git reset, что в полном виде git reset –mixed HEAD), что является аналогом git restore .

Git reset <имя файла> - команда противоположная add – удаляет указанный файл из индекса.

Git reset <хэш коммита> <имя файла> - подтянет версию указанного файла из указанного коммита только в индекс (файл будет помечен как modified как в рабочей, так и в индексе) в отличие от git checkout <хэш коммита> <имя файла>, который добавляет в рабочую и в индекс.

2 других (основаны на --hard):

--merge –

Отмена слияния: отмена конфликтного merge, аналог merge –abort – откатывает процесс слияния, сохраняет незакомиченные изменения в рабочей директории.

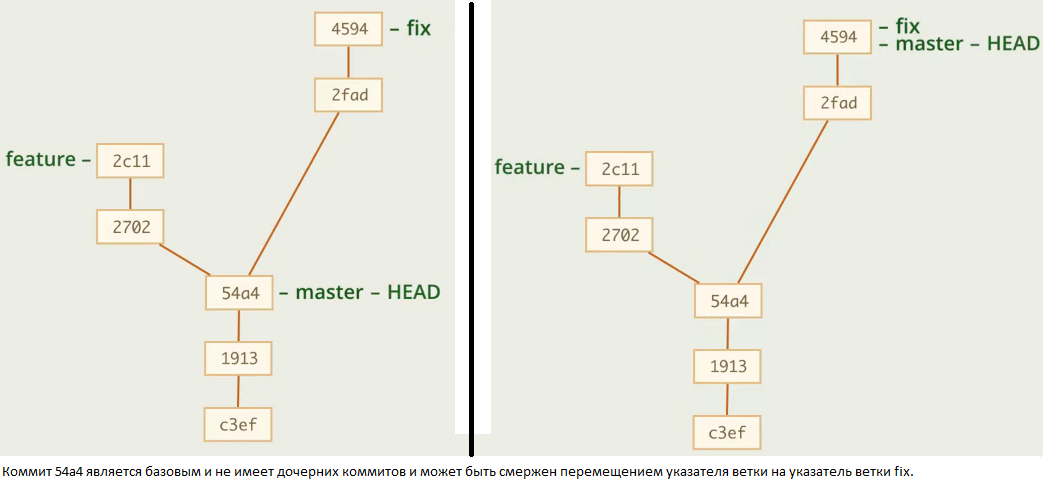
Обычное использование: Очищает индекс – не переносит изменения из индекса в рабочую директорию, а именно очищает. Изменения, которые находились в рабочей директории сохраняются.

--keep – работает как mixed (сбрасывает индекс), но отличается поведение для рабочей директории – если изменения в рабочей директории конфликтуют с изменениями в указанном коммите (файл в коммите изменялся), то reset не будет выполнен.

**Виды merge:**

1) Fast Forward (Перемотка) – происходит, если у исходной ветки отсутствуют коммиты после базового (общий коммит для нескольких веток), в таком случае, указатель исходной ветки перемещается к указателю вливаемой ветки, при этом не создается коммита слияния.

Такой способ слияния может быть неудобен, т.к. после слияния сложно определить с какого места ветка была смержена (понадобится в случае необходимости отката merge). Флаг –no-ff не будет использовать FF мерж, а воспользуется true merge, то есть создаст визуально различимые базовый коммит и коммит слияния.



2) True merge (Истинное слияние) – обе ветки имеют коммиты после базового, поэтому нельзя просто передвинуть один из указателей ветки так как обе ветки точно содержат различные версии файлов. В таком случае сливаемые ветки объединяются посредством одной из стратегий слияния.

Общий алгоритм – находится базовый коммит для этих веток (ближайший общий коммит для этих веток, может быть получен с помощью git merge-based <Ветка> <Ветка>) – он является отправной точкой для анализа изменений (происходили ли они или нет), затем изменения одной ветки накладываются на версию файла в базовом коммите, после на этот же файл накладываются изменения из другой ветки, что может привести к 3м сценариям:

Отсутствие изменений в обеих ветках – берется версия базового коммита  
Изменение произведено только в 1 из веток – берется эта версия  
Изменение присутствует в обеих ветках – возникновение состояния прерванного слияния или конфликта.

\*Под изменением понимается версия строки, а не изменение в целом.

**Стратегии слияния –** конфигурируются флагом –s <название стратегии>

Resolve – Находится самый ранний общий предок и сравниваются изменения с коммитами веток по общему алгоритму.

Минус этого подхода в том, что если в процессе разработки уже были merge с конфликтами, то будущие merge по такой же стратегии потребуют снова разрешения этого конфликта.

Recursive (стратегия по умолчанию) – В качестве базы для слияния используется виртуальный предок, который формируется за счет поиска всех общих предков и слияния их между собой, далее работает по общему алгоритму. Не имеет недостатков Resolve стратегии.

Octopus – используется для слияния сразу нескольких веток и созданием коммита-слияния более чем с 2мя родителями. Альтернативный подход – мержить каждую ветку по очереди, но тогда создадутся несколько коммитов слияния.

Ours – Формальное объединение истории при помощи коммитов слияния при полном игнорировании содержимого сторонней ветки (merge коммит по содержанию будет соответствовать предыдущему коммиту, но иметь двух родителей)

Subtree – используется для объединения подпроектов или для слияния подпроекта с основным проектом, сохраняя историю и структуру изменений.

**Прерванное слияние или конфликт** – невозможность автоматического добавления изменений версий файлов из разных веток (помечаются как both modified). Технически это состояние выражено наличием в индексе 3х версий файла (2х веток и базовой), изменения в которых конфликтуют между собой. Такое состояние требует ручного редактирование файла, который содержит конфликт (в него будут добавлены обе версии изменений).

Переключаться между версиями файлов при конфликте можно с помощью   
git checkout –ours/--theirs <Имя файла>  
git checkout –merge <Имя файла>

Состояние прерванного слияния может быть разрешено либо разрешением конфликта и последующим вызовом команды merge --continue или git commit, что приведет к завершению слияния и создания коммита слияния, который будет иметь двух родителей. Или окончательным прерыванием коммита путем вызова git reset –hard (откатит merge, также удалить все незакомиченные изменения в рабочей директории и индексе) либо git reset –merge/ git merge –abort (откатит merge, но оставит изменения, не относящиеся к merge – останутся изменения, которые не были закоммичены и вызвался merge).

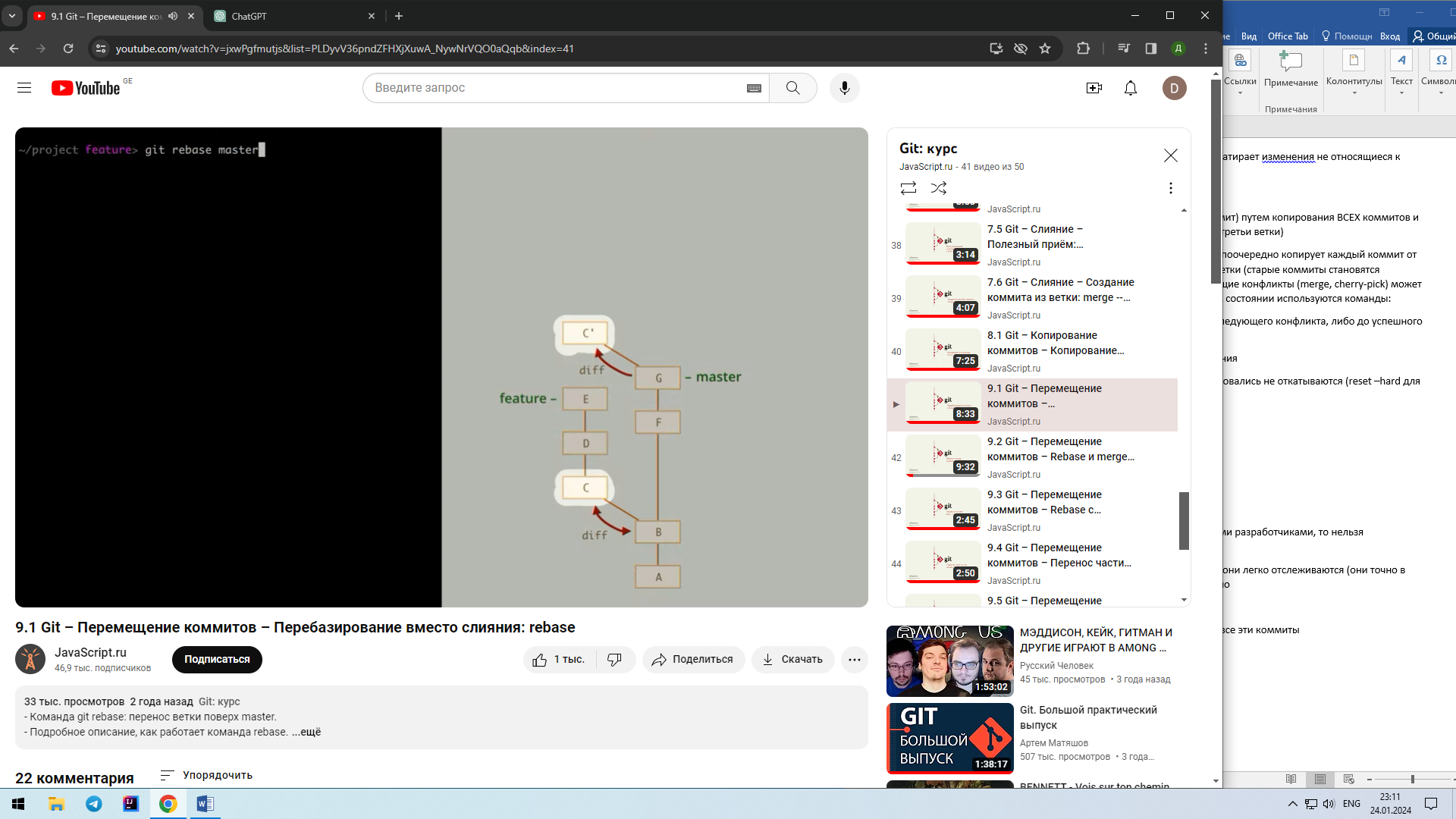
Помимо конфликтов при слиянии, есть так же **семантические конфликты** – для git файлы технически совместимы и могут быть смержены автоматически, но они несовместимы по смыслу (нарушается логика программы или просто код не скомпилируется)

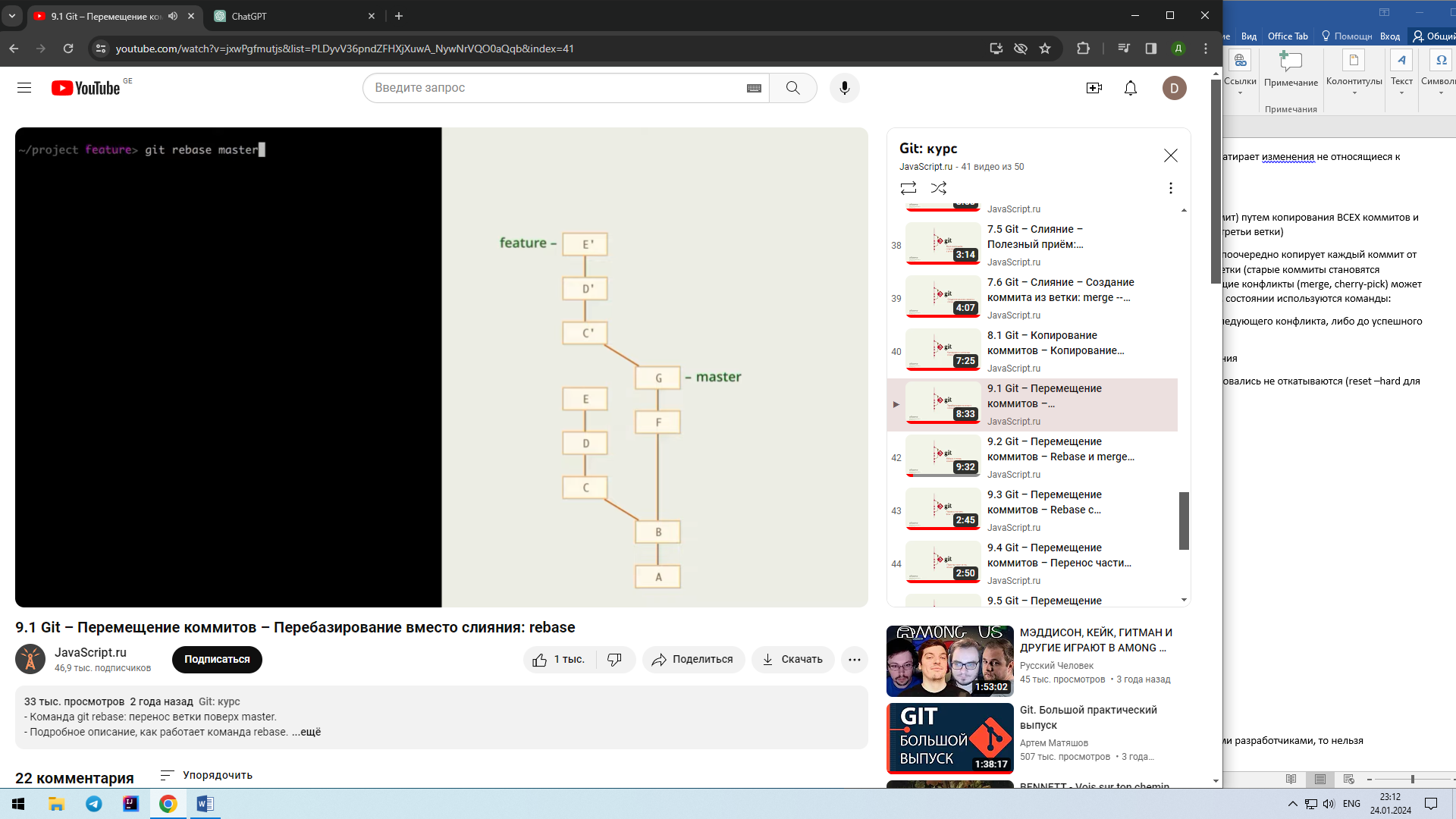
*Флаг -–no-commit* – такое слияние подразумевает добавление изменений в рабочую директорию и индекс (предварительный просмотр), но не создает коммит слияния сразу. Подтверждение этих изменений создает коммит слияния.

*git merge –squash <ветка>* такая команда приводит к созданию сжатого коммита – изменения из указанной ветки будут перемещены в рабочую директорию и индекс текущей ветки без создания коммита слияния, что приводит к тому, что история изменений не будет содержать прямой связи между ветками (используется в случае, если нужен результат разработки, но не интересна история). При таком коммите не создается указатель MERGE\_HEAD (хранит ссылки на коммиты родители), поэтому невозможно использование команд merge –abort/continue для завершения процесса «слияния» - использовать git reset –merge/ git commit.

**git rebase** – изменяет базовый коммит для текущей ветки путем поочередного применения всех коммитов на новый базовый коммит (по умолчанию указатель ветки).

Общий вид: git rebase <новый base коммит> по умолчанию будет перемещена вся текущая ветка, но только те коммиты, которые не присутствуют в ветке, на которую перебазируют.

Пример git rebase master (находимся на feature)



Rebase также подразумевает возможность возникновения конфликтов, выход из которого также осуществляется с помощью флагов:

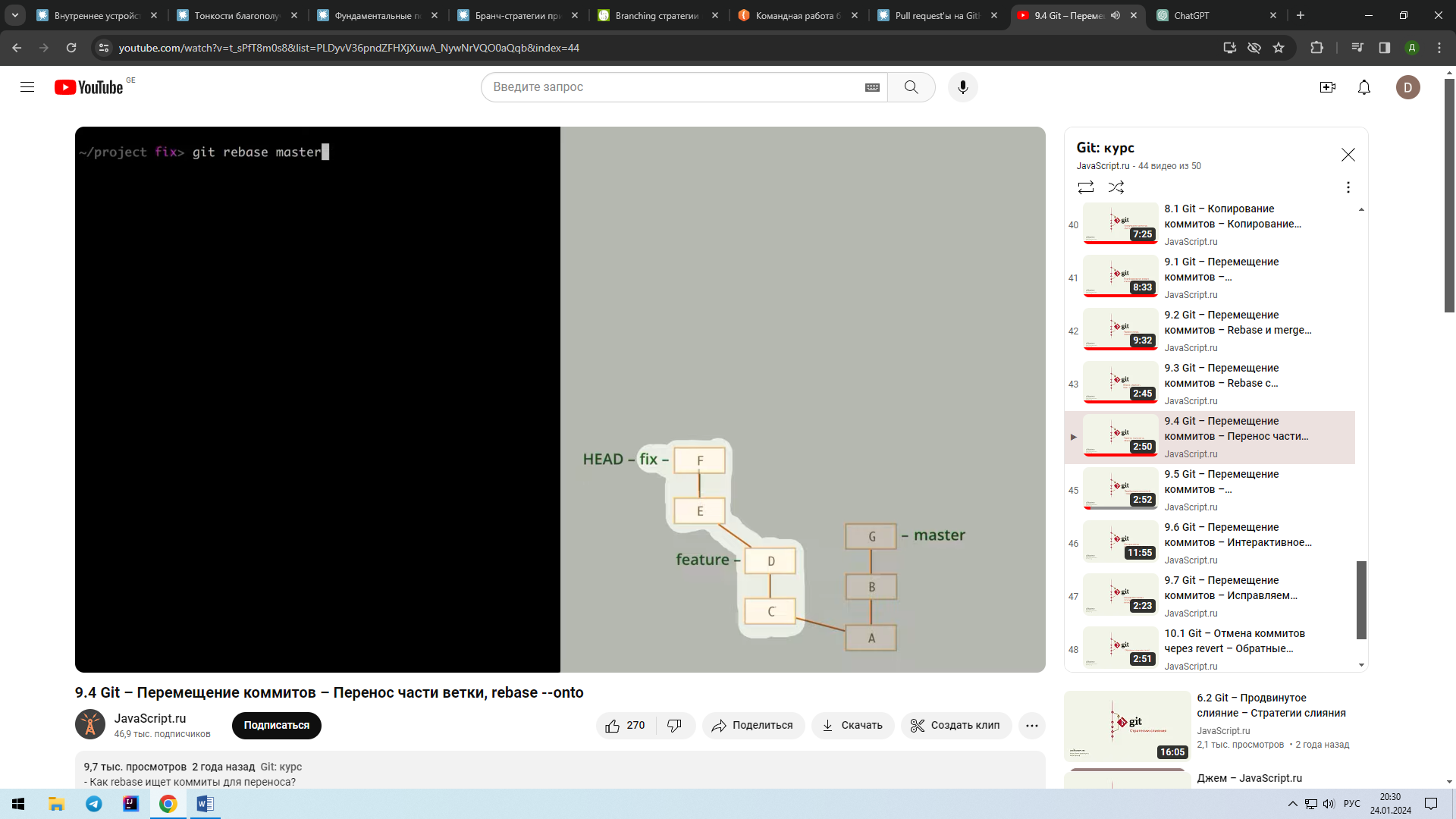
--continue – продолжает перебазирование с места конфликта до следующего конфликта, либо до успешного завершения.

--abort – возвращает исходное состояние до начала перебазирования

--quit – прерывает процесс, коммиты, которые успешно перебазировались не откатываются (reset –hard для этой команды работает аналогично quit)

**Особенности:**

*Перенос части ветки*

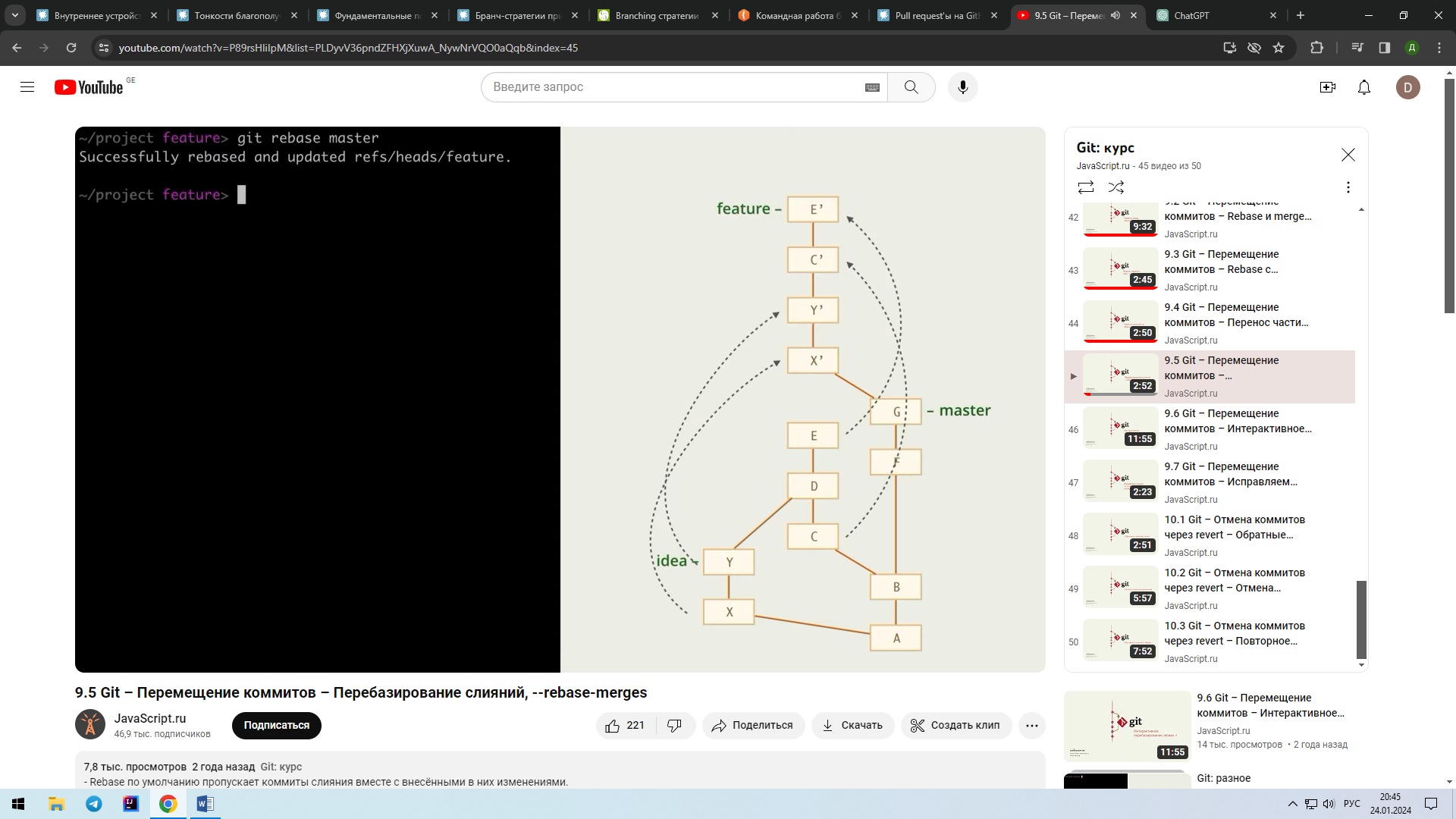
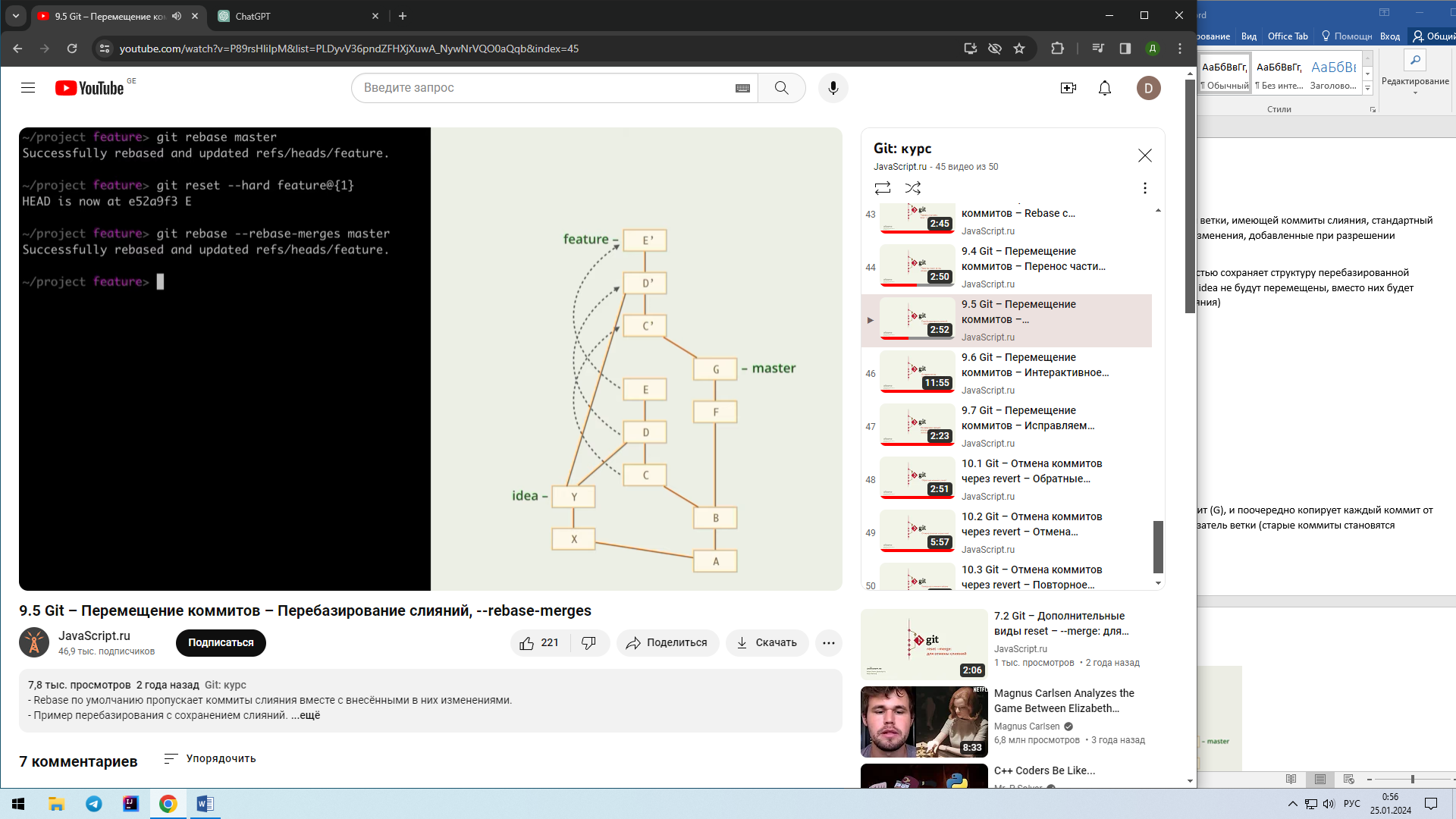
В данном случае, базовый коммит – A, при вызове rebase c ветки fix он перенесет все коммиты от A до HEAD, то есть ветка feature тоже будет перебазирована.

Так же перебазируются только те коммиты, которых нет в master (если бы ветка feature была смержена в master, то ее коммиты бы не включились в rebase)

Если необходимо перенести только ветку fix, то нужно использовать флаг –onto <новый базовый коммит> <точка начала переноса>.

Для примера команда будет выглядеть git rebase -- onto master feature (Находимся на fix. Коммиты E, F будут перебазированы на G).

*Перебазирование при наличии коммитов слияния*

Изображены результаты исполнения git rebase master (перебазированию подлежат все коммиты, которых нет в master).

При перебазировании ветки, имеющей коммиты слияния, стандартный rebase их пропустит (изменения, добавленные при разрешении конфликта, пропадут).

–rebase-merges полностью сохраняет структуру перебазированной ветки. Коммиты ветки idea не будут перемещены, вместо них будет добавлен коммит слияния, но конфликты в коммите слияния придется решать повторно из-за того, что rebase по сути пересоздает все коммиты. Для повторных конфликтов можно использовать rerere.

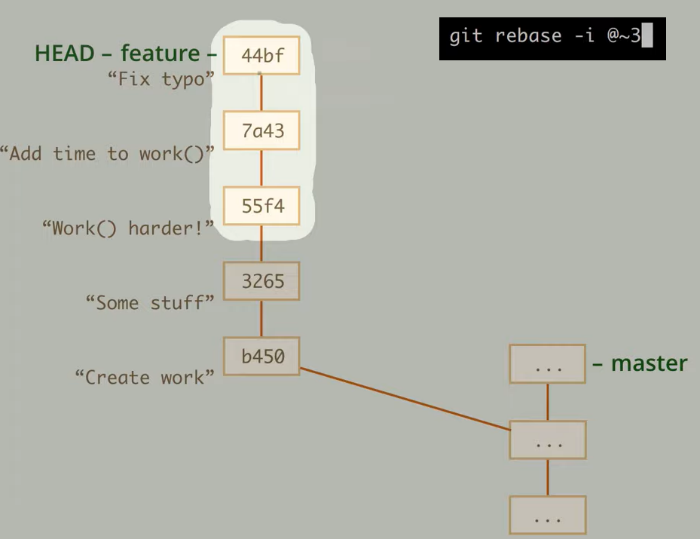
reuse recorded resolution (rerere) – механизм для запоминания ранее разрешенных конфликтов и их повторного автоматического применения в случае его повторения. git config rerere.enabled true для включения, после этого он начнет запоминать все разрешенные конфликты. Rerere –train запустит обучение механизма (пройдет по дереву и запомнит все ранее решенные конфликты). [Видео.](https://www.youtube.com/watch?v=8HxTHPkdedA&list=PLDyvV36pndZEB7kWWocU4QSn-G78LoaEE&index=5)

*Флаг –x* “исполняемый файл” – запускает указанный файл для каждого коммита после его успешного перебазирования.

*Флаг –I* – позволяет гибко управлять процессом перебазирования, а именно явно задавать действия, которые необходимо совершить с каждым перебазируемым коммитом (изменение его истории), а именно:

Изменение последовательности коммитов с помощью определения их порядка в интерактивном окне.  
Изменение сообщения коммита (reword)  
Редактирование содержимого коммита (edit)  
Объединение коммитов с сохранением или удалением сообщения (squash/fixup)  
Удаление коммита (drop)

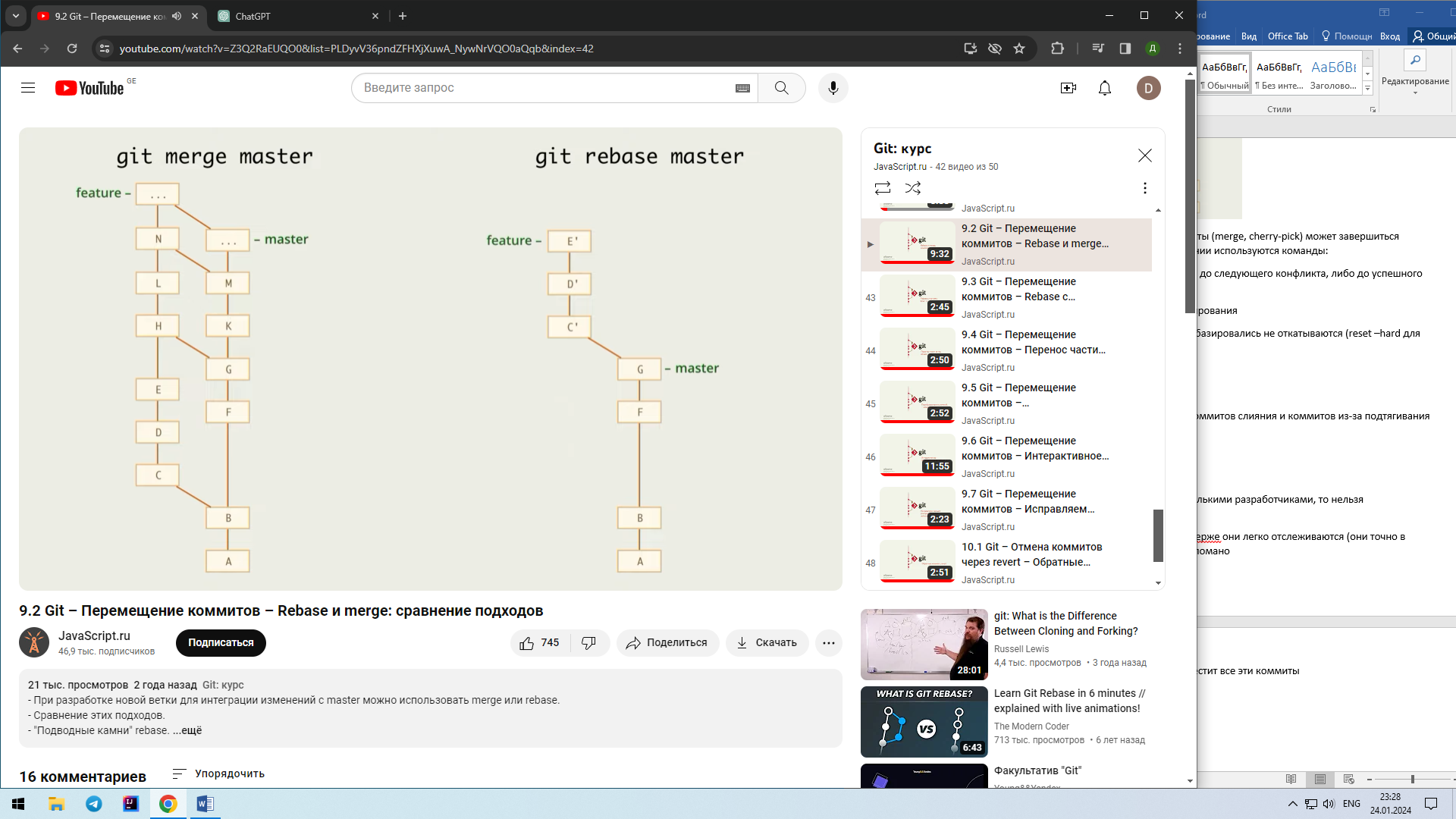
Продолжение перебазирования после окончания редактирования коммита осуществляется с помощью git rebase –continue, просмотр оставшихся задач для перебазирования git rebase –edit-todo.

Интерактивное перебазирование не обязательно используется для изменения базового коммита ветки, оно также может быть использовано для редактирования истории текущей ветки путем обозначения соответствующего диапазона при вызове.

При необходимости изменить коммит, который находится в середине ветки можно вручную создать этот коммит и в интерактивном режиме произвести редактирование (изменить порядок и удалить ненужный) либо воспользоваться автоматической заменой таких коммитов: создать нужный коммит с флагом –fixup=<Хэш коммита> и вызвать rebase с флагом –autosquash, при таком вызове коммиты будут заменены автоматически.

**Merge или rebase:**

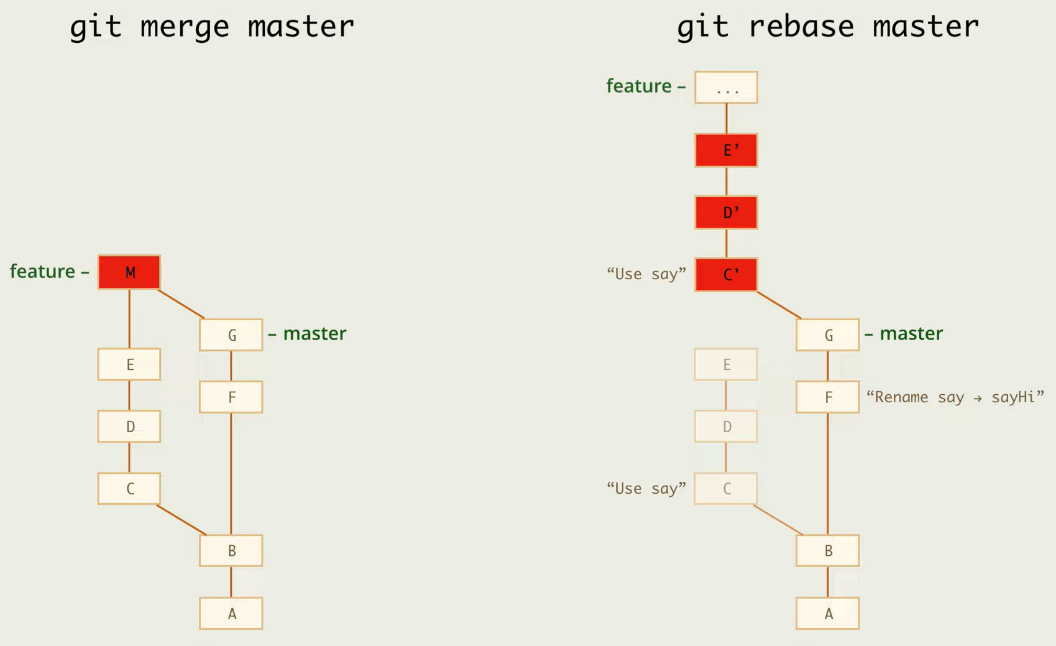
Rebase – Упрощает чтение истории разработки (за счет отсутствия коммитов слияния для подтягивания изменений из другой ветки)



Минусы:

Нельзя перебазировать публичные ветки из-за последующих проблем с конфликтами у разработчиков, использующих эту ветку ([подробнее](https://habr.com/ru/companies/otus/articles/352640/))

При наличии семантических конфликтов «сломанными» окажутся все перебазированные коммиты, что можно решить с помощью fix-commit, но это нарушит принцип, который гласит, что каждый коммит должен отражать рабочее состояния кода. При использовании всегда известна точка, в которой произошел семантический конфликт.



**Стратегии ветвления:**

**Centralized strategy** *–* вся разработка осуществляется только в одной основной ветке.

**Feature-branch workflow** – новый функционал разрабатывается в ветках feature, которые отходят от основной ветки, сливаются изменения, соответственно, в основную ветку.

[**Git Flow**](https://www.youtube.com/watch?v=umiT0yIsSrc) – Основные ветки:

master - находится стабильная версия кода, готовая к выпуску в продакшн.   
develop - используется для ведения разработки, содержит последние изменения, готовые к интеграции в проект.

Дополнительные ветки:

feature – для разработки нового функционала, создается и мержатся в develop.  
hotfix – для быстрого исправления багов в проекте. Создаются из ветки master, исправления мержатся в master и develop.  
release – для подготовки выпуска новой версии проекта (доп. Тесты, тэги и тд). Из ветки develop изменения мержатся в нее, затем в master и develop.

[**Trunk – based**](https://www.youtube.com/watch?v=31_IdSFvQrI) – использует одну основную ветку (main/master/trunk), остальные ветки короткоживущие (не более 2х дней), для быстрой доставки изменений.

Feature ветки – ведется разработка нового функционала, атомарные – функционал, разрабатываемый в ветке разбивается на небольшие подзадачи и сразу вмерживаются в основную ветку.

Концепция feature flags – включение/выключение фичей по ключу, необходима из-за

За счет небольших изменений код ревьюится за несколько минут и быстрее поступает в продакшн

Для доставки изменений различные стратегии может использоваться централизованный контроль за слиянием посредством концепций *Integration manager или Dictator and Lieutenants.*

*Integration manager* – Подразумевает blessed repository, который содержит финальную версию кода, пушить в который может только один человек (integration manager).

*Dictator and Lieutenants* – Похожа на integration manager, в данном подходе несколько менеджеров, которые могут пушить код, но они работают на разных уровнях, лейтенанты осуществляют «предпроверку», с которой уже работает диктатор и пушит в blessed repository

*Forking Workflow –* стратегия для open-source разработки, используется в github. Представляет оригинальный репозиторий, изменения в который может вносить только владелец этого репозитория. Сторонние разработчики (не владельцы), для предложения изменений в репозиторий, копируют этот репозиторий (fork) и вносят в него изменения, а затем предлагают эти изменения владельцу репозитория (pull request/perge request).

\*Trunk-based и git Flow являются базовыми стратегиями, на их основе базируются остальные (различные flow, ops и другие).

\*Теория и визуализация команд\* [№1](https://www.youtube.com/playlist?list=PLDyvV36pndZFHXjXuwA_NywNrVQO0aQqb)/ [№2](https://www.youtube.com/playlist?list=PLDyvV36pndZEB7kWWocU4QSn-G78LoaEE)

\*Источники\*: [Общая информация](https://habr.com/ru/companies/badoo/articles/163853/), [Стратегии слияния](https://habr.com/ru/articles/195674/), Бранч-стратегии: [1](https://habr.com/ru/articles/765264/), [2](https://habr.com/ru/companies/itglobalcom/articles/535524/), [pull request](https://habr.com/ru/articles/125999/).