Система контроля версий (VCS) – система, которая фиксирует изменения в пакете в виде версий, хранит историю изменений и при необходимости есть возможность вернуться к любой версии.

*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.

***Статусы***

*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.

|  |  |
| --- | --- |
| Untracked | Новый файл, про который в репозитории еще не записей |
| Staged | Файлы, подготовленные для коммита, перемещенные в зону индекс |
| Unmodified | Файлы, состояние которых в рабочей директории и репозитории равны (то есть они синхронизированы) |
| Modified | Файл, в рабочей директории, который изменил свое состояние по отношению в Репозиторию |
| Commited | Текущее снимок файла в рабочей директории соответствует репозиторию |

***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

***Объекты git:***

|  |  |
| --- | --- |
| Blob | Контейнер для бинарных данных, то есть этот объект является сжатым содержанием этого файла без метаданных файла |
| Tree | Аналог директории в репозитории, то есть содержит информацию о файлах (блобах) и других директориях (деревьях) |
| Commit | Сохраненное состояние репозитория в определенный момент времени. 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 родительский коммит (все коммиты связаны между собой посредством этого механизма, как односвязный список), ссылка на корневое дерево.  При создании нового коммита всегда создается новое корневое дерево, которое содержит ссылки на другие объекты git, при этом, при создании нового коммита, объекты git которое не были изменены не будут пересозданы для данного коммита, а просто используются ссылки на эти объекты из предыдущих коммитов.  Коммит должен быть атомарным и консистентным – Коммит должен отмечать только один функционал и быть логически завершенным (Commit early. Commit often.) |
| Annotated Tag | Метка обозначающая важный события в репозитории |

SHA-1 Хэш – Хэш функция, значение которой весит 20 байт, содержит 40 символов в 16-ричной системе. Хэш-сумма вычисляется у каждого объекта git в зависимости от его содержимого, т.е. хэш дерева будет изменен, если изменился блоб, который содержался в этом дереве. Используется в качестве имен объектов и идентификаторов для ссылок.

Принцип работы – При создании коммита вначале формируется корневое дерево – блобы и деревья, которые не были изменены подтягиваются из предыдущих коммитов в виде ссылок, остальные блобы и деревья, хэш которых был изменен, пересоздаются и вычисляются их хэш суммы и ссылки передаются в корневое дерево. Далее коммит связывается с корневым деревом и с предыдущим коммитом.

**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 | Просто создать пустой файл |

**Команды GIT – все команды начинаются с git**

**Git <команда> -h –** выводит все флаги этой команды и короткое описание

**Git help <команда>** - выводит описание команды и так же ее опции

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Описание*** | ***Флаги*** | ***Описание*** |
| **Инициализация Git Репозитория** | | | |
| init | Создание репозитория | | |
| ***Команды для настроек и отображения информации*** | | | |
| config | Установка параметров и конфигураций.  Задается уровень видимости конфигурации:  system – все пользователи системы;  global – конкретный пользователь;  local – текущий репозиторий;  Пример:  git config –global set.name “some name” | --list | список всех настроек Git |
| 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 изменения | <Хэш коммита> | Отображает инфу по указанному коммиту, если не указывать коммит, то выведет HEAD |
| <Коммит>:<Имя> | Выводит не отличия, а полностью файл в текущем коммите |
| status | Проверка состояния индекса и рабочей директории |  |  |
| 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 |
| -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 (использует регулярные выражения) |
| -G | Ищет совпадения в файлах |
| reflog | Команда, для обращения к файлу с записями об изменениях ссылок (.git/logs/) применяется к HEAD и названиям веток.  Рефлоги находятся только локально, их нет в GitHub |  | С помощью данных, имеющихся в reflog, можно переключаться на коммиты (например идентификатор 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 | 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) | 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 | Коммитит ВСЕ файлы, сразу из рабочей директории (работает только для модифицированных файлов) |
| --amend | 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 с текущими изменениями |
| --author | Установить автора для коммита – метаданные коммита включают автора файла и коммитера |
|  |  | --fixup=<коммит> | Создает коммит-заплатку, с отредактированным заголовком (добавляет «fixup!») |
| clean | Для удаления Untraced файлов | -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 | Сохраняет незакоммиченные изменения и архивирует, позволяет переключаться между ветками при наличии незакомиченных изменений  Не привязывает заархивированные изменения к определенной ветке! | pop | Бросает незакомиченные изменения в текущий HEAD |
| restore | Сбрасывает состояние рабочей директории до состояния HEAD | -- staged | Перебрасывает файлы из индекса обратно в рабочую директорию |
| 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 | Не удаляет файл из рабочей директории |
| -r или --recursive | Рекурсивно удаляет папку (сначала содержимое, потом папку) |
| **Команды для перемещения указателей, управления коммитами** | | | |
| branch | <Имя ветки> Создает новую ветку от текущей позиции HEAD  <Без параметров> Показывает все локальные ветки (не показывает ветки удаленного репозитория) | -a | Показывает все существующие ветки, включая ветки на удаленном репозитории |
| -v | Тоже самое, только с хэш-суммой |
| -d | Удаляет ветку |
| -D | Удаляет ветку независимо от того, были закоммичены изменения или нет |
| -f <Имя><Коммит> | Перемещает указатель ветки на указанный коммит (нельзя выполнить на этой же ветке – нужно переключиться на другую) |
| checkout | Перемещает указатель HEAD и приводит рабочую директорию к состоянию этого коммита. Можно использовать как на ветке, так и на отдельном коммите  Команда не позволяет переключиться на другую ветку, если в текущей есть незакомиченные изменения. Исключение, если сделанные изменения будут добавлены в индекс и будут соответствовать текущему состоянию ветки, на которую переключились, при этом затрутся изменения в ветке, с которой переключились | -b <Имя> | Создает ветку <Имя> и перемещает на нее HEAD (не нужно вызывать сначала branch а потом checkout) |
| -b <Хэш коммита> | Создаст ветку от указанного коммита |
| <Имя файла> | Возвращает рабочую директорию к состоянию индекса (если там есть изменения) иначе синхронизируется с репозиторием |
| -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) |
| <Коммит><Имя файла> | Подтягивает состояние указанного файла в текущий указатель HEAD – нужен в ситуации, если нужна какая-то прошлая версия файла. Автоматически добавляет в индекс |
| tag | Создание тэга, который указывает на какой-то важный коммит (нельзя перемещать)  Тэги не пушатся вместе с коммитами на сервер. | <Имя> | Легковесный тэг |
| –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  --soft  --mixed |  |
| cherry-pick | Копирует заданные коммиты и применяет его к текущему HEAD, коммиты могут быть перечислены посредством перечисления или диапазоном (master..feature) | -x | Добавляет строку с информацией о том, откуда он был скопирован |
| -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 | Коммитит изменения |
| merge | [\*Слияние веток\*](#merge) Указатель Head на той ветке в которую сливаем изменения, <Имя ветки> из которой сливаем |  |  |
| rebase | [\*Перебазирование\*](#rebase) |  |  |
| revert | Создает коммит с противоположными указанному коммиту изменениями (коммит отмены). Задается один коммит, либо диапазон. | -m <Родитель> | Нужен для коммитов слияния, для указания, изменения какого имя родителя нужно отменить. |
| **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | | | |
| ^ | 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) | | | |

**Указатели:**

1) Head – указатель на текущую ветку или на конкретный коммит, от этого указателя всегда создаются новые коммиты

2) Указатель на ветку, он указывает на последний коммит в ветке – в виде имени ветки.

3) Ветка слежения - указатели веток, связанные с удаленным репозиторием - 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).

**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 в каком месте сливаемая ветка была создана и смержена. 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. В таком случае сливаемые ветки объединяются посредством одной из [стратегий слияния](#Слияние). Слияние может закончится либо созданием коммита слияния (коммит, содержащий изменения из двух веток и имеющий двух предков), либо возникновением конфликта – в обоих версиях файла изменения не могут быть объединены автоматически (помечаются как both modified), разрешение такого конфликта включает в себя ручное редактирование конфликтующих файлов и их последующий коммит (merge --continue или git commit) для завершения слияния. Переключаться между версиями файлов при конфликте можно с помощью git checkout –ours/--theirs <Имя файла>

Отмена незавершенного коммита – либо reset –hard (откатит merge и зачистит изменения в рабочей и индексе), либо merge –abort / reset –merge (откатит merge, но изменения, не относящиеся к merge, оставит)

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)

Избежать такой ситуации поможет флаг –no-commit, который даже в случае успешного слияния не создаст коммит слияния, далее по этому состоянию прогнать тесты, для проверки корректности состояния.

**Стратегии слияния**

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 (Работает как Resolve, но за счет наличия виртуального предка не имеет его минусов)

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 (Мерж коммит по содержанию будет соответствовать предыдущему коммиту, но иметь двух родителей)

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

1) Файл не изменился ни в одной ветке – записывается в merge-коммит

2) Файл изменился в одной из веток – берется версия измененного

3)Файл изменился в обеих ветках – конфликт (прерванное слияние), надо решать руками

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.

subtree

**Стратегии ветвления**

*Centralized strategy –* Все разработчики работают только в master.

*Feature-branch workflow* – Новый функционал разрабатывается в feature, все ветки feature бранчатся от master и вмерживаются в master.

GitHub Flow – похожа на *Feature-branch workflow.* Изменения в master вносятся через pull request, код в master всегда должен быть готов к релизу

*Git Flow* – 2 главные ветки (master, develop) – в ветке мастер находится версия кода, которая используется в продакшне. Develop содержит весь код ветки master + новые функции (для дополнительного тестирования)

Feature ветки – для разработки новых функциональностей, создается и вмерживается в develop

Hotfix – для быстрого исправления багов в проекте. Создаются из ветки master, вмерживается в master и develop

Release – для подготовки выпуска новой версии проекта (доп. Тесты, тэги и тд). Создается из develop, вмерживается в master и develop

*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. Есть оригинальный репозиторий, с него другие разработчики снимают копию (fork) и изменения, внесенные в этот репозиторий, отправляются владельцу оригинального репозитория посредством pull request.

Trunk

***Команда reset***

Достижимость коммита – Коммит считается неодстижимым, если он не включен ни в одну из существующих веток, такие коммиты с течением времени (30 дней) будут удалены GC.

В общем смысле команда перемещает указатель ветки и указатель HEAD на указанный коммит, тем самым как бы откатывая репозиторий назад, отменяя уже закомиченные изменения, они остаются в репозитории, но при этом будут удалены по умолчанию через 30 дней, дальнейшие коммиты будут продолжаться с указателя ветки.

3 основных флага:

--hard откатывает рабочую директорию до состояния коммита на котором указатель, при этом полностью зачищает рабочую зону и индекс ото всех изменений, которые лежали там до этой команды!

Если коммит не указан, то подразумевается HEAD, тогда флаг перемещен не будет, а просто затрутся изменения в рабочей директории и индексе Git reset --hard

--soft Рабочая директория и индекс остается без изменений, то есть внесенные, но незакомиченные изменения не затираются. При этом разница между предыдущим коммитом и настоящим добавляются в индекс

--mixed – является режимом по умолчанию (можно не писать) сбрасывает состояние индекса, но рабочая директория остается нетронутой. Разница между коммитами добавляются в рабочую директорию.

Часто применяется для очистки индекса: git reset или git reset –mixed HEAD (тоже самое)

Git reset <Коммит> <Имя файла> - обновляет состояние файла до текущего коммита и добавляет его в индекс – аналог команды git checkout <Коммит> <Имя файла> - эта добавляет и в индекс и в рабочку

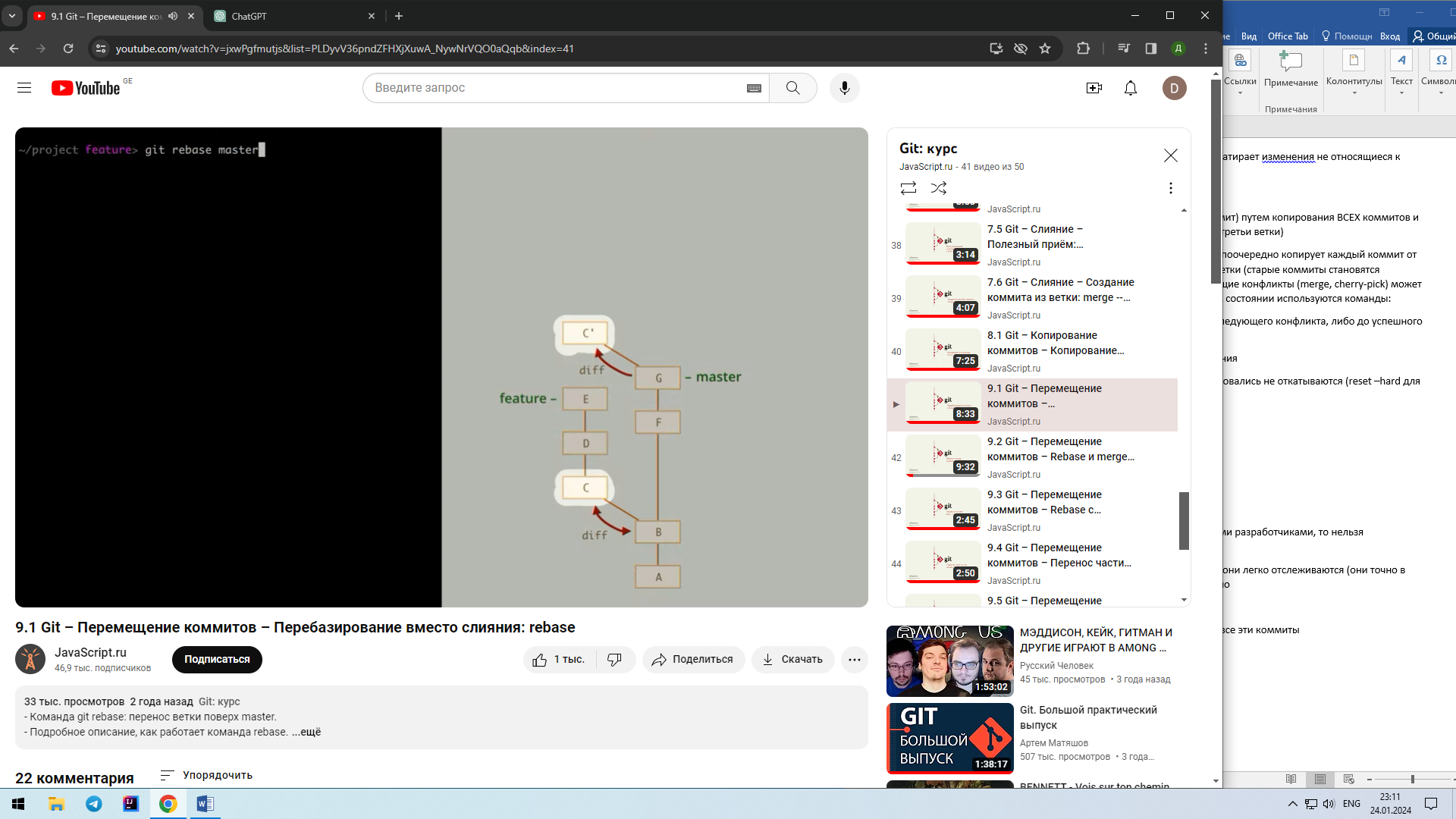
2 других:

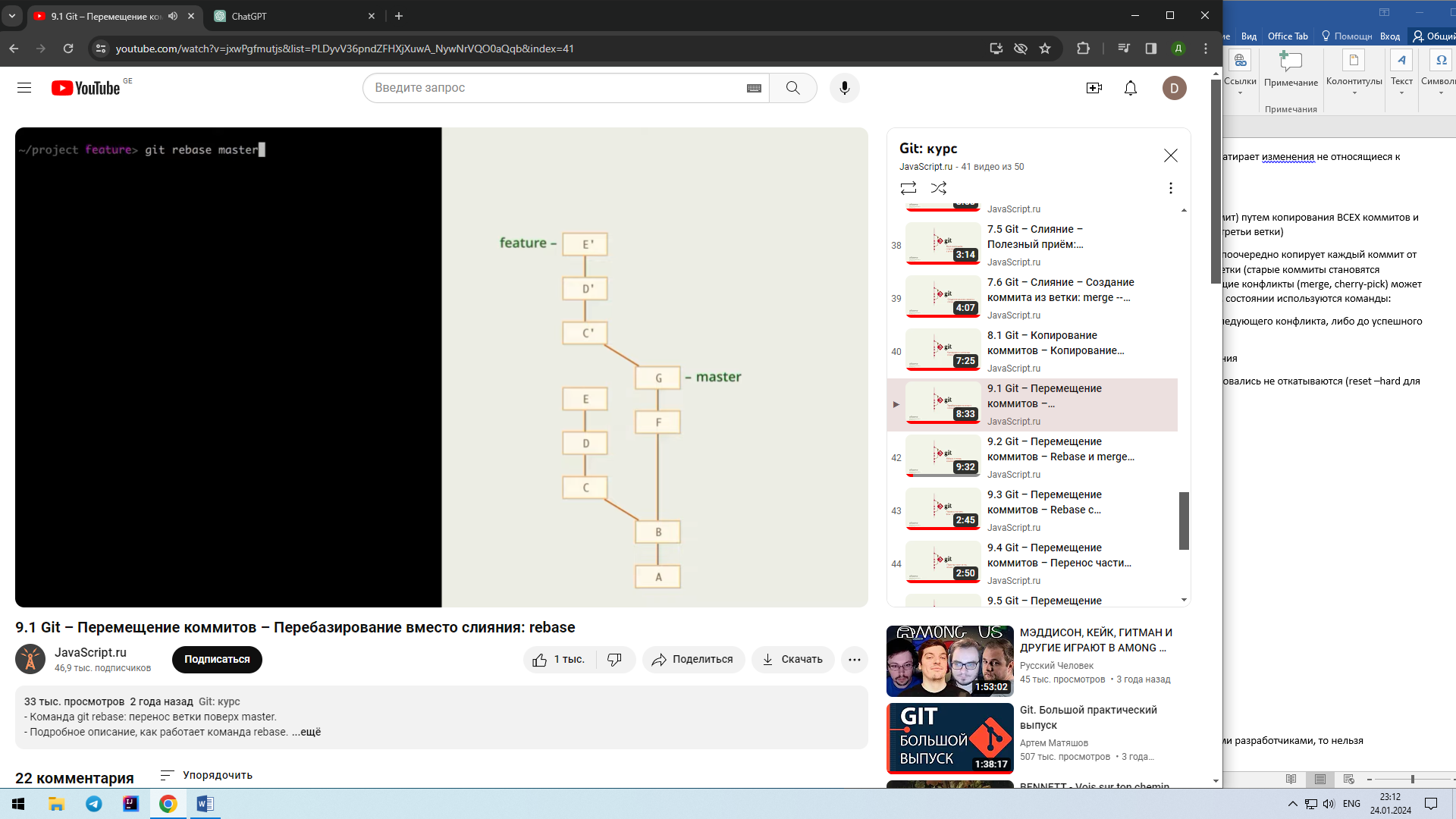
--merge – отмена незаконченного merge, аналог merge –abort (не затирает изменения, не относящиеся к merge)

--keep – оставляет незакоммиченные изменения в рабочей директории при условии, что фал не менялся между коммитами

**git rebase** – изменяет базовый коммит (им становится другой коммит) путем копирования ВСЕХ коммитов перебазируемой ветки (включая предков) и перемещения на них соответствующего указателя ветки.

Принцип работы: Перемещает HEAD на новый базовый коммит (G), и поочередно копирует каждый коммит от предыдущего базового коммита (B), в конце перемещает указатель ветки (старые коммиты становятся недосягаемыми).





Как и все предыдущие команды (merge, cherry-pick), подразумевающие конфликты, может завершиться успешно или перейти в состояние конфликта, в таком состоянии используются команды:

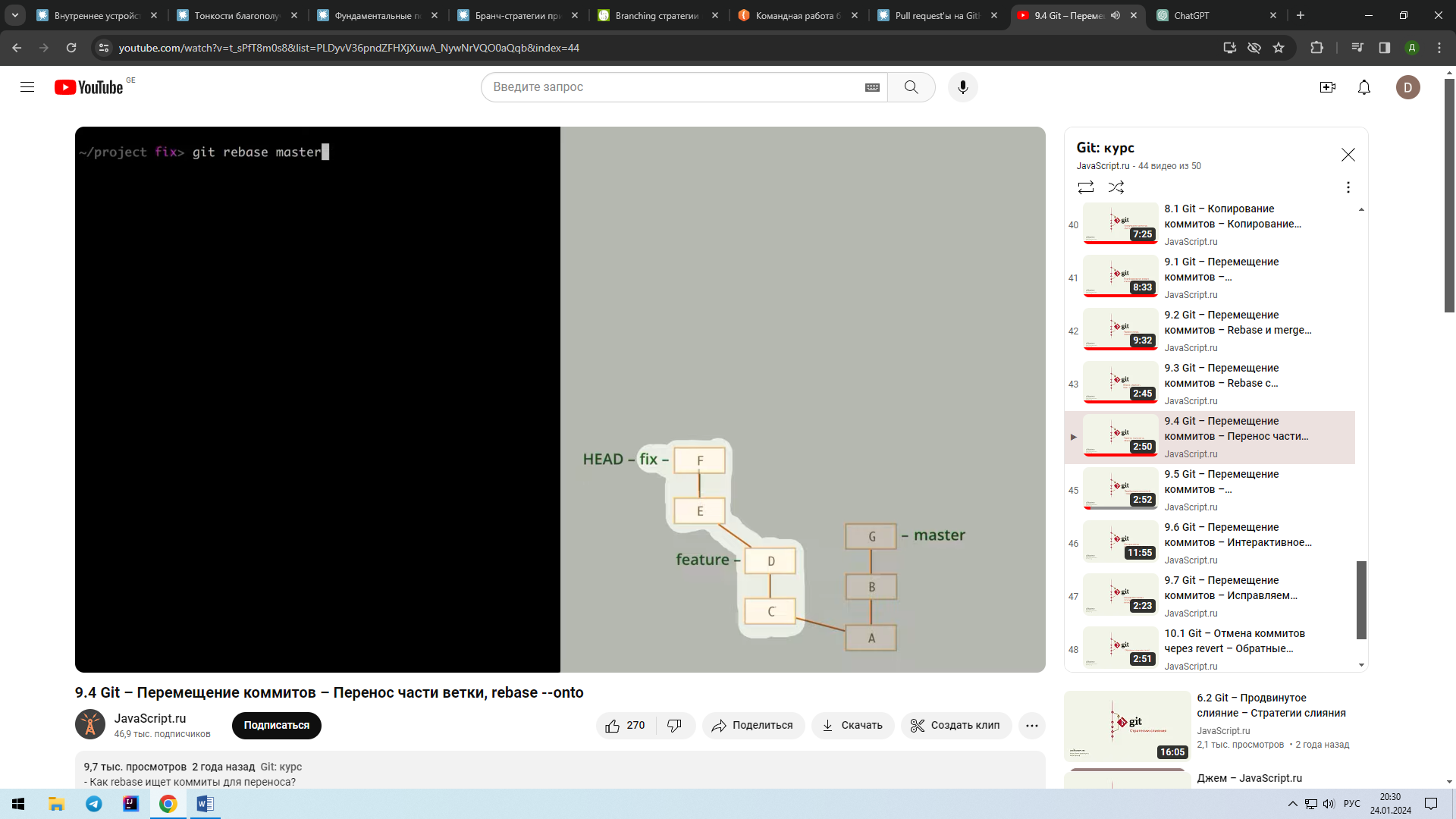
--continue – продолжает перебазирование с места конфликта до следующего конфликта, либо до успешного завершения.

--abort – возвращает исходное состояние до начала перебазирования

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

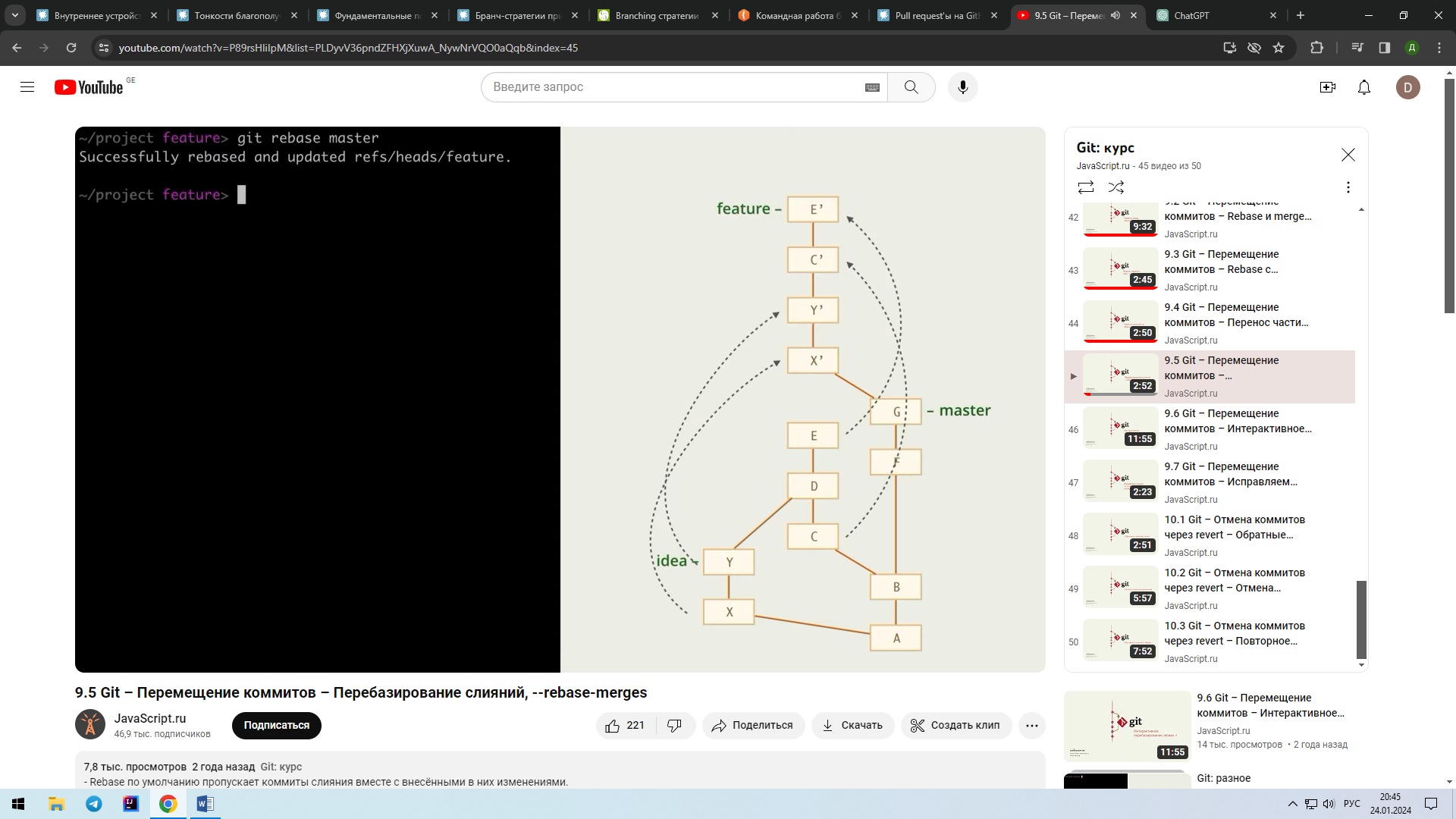
Особенности:

*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.

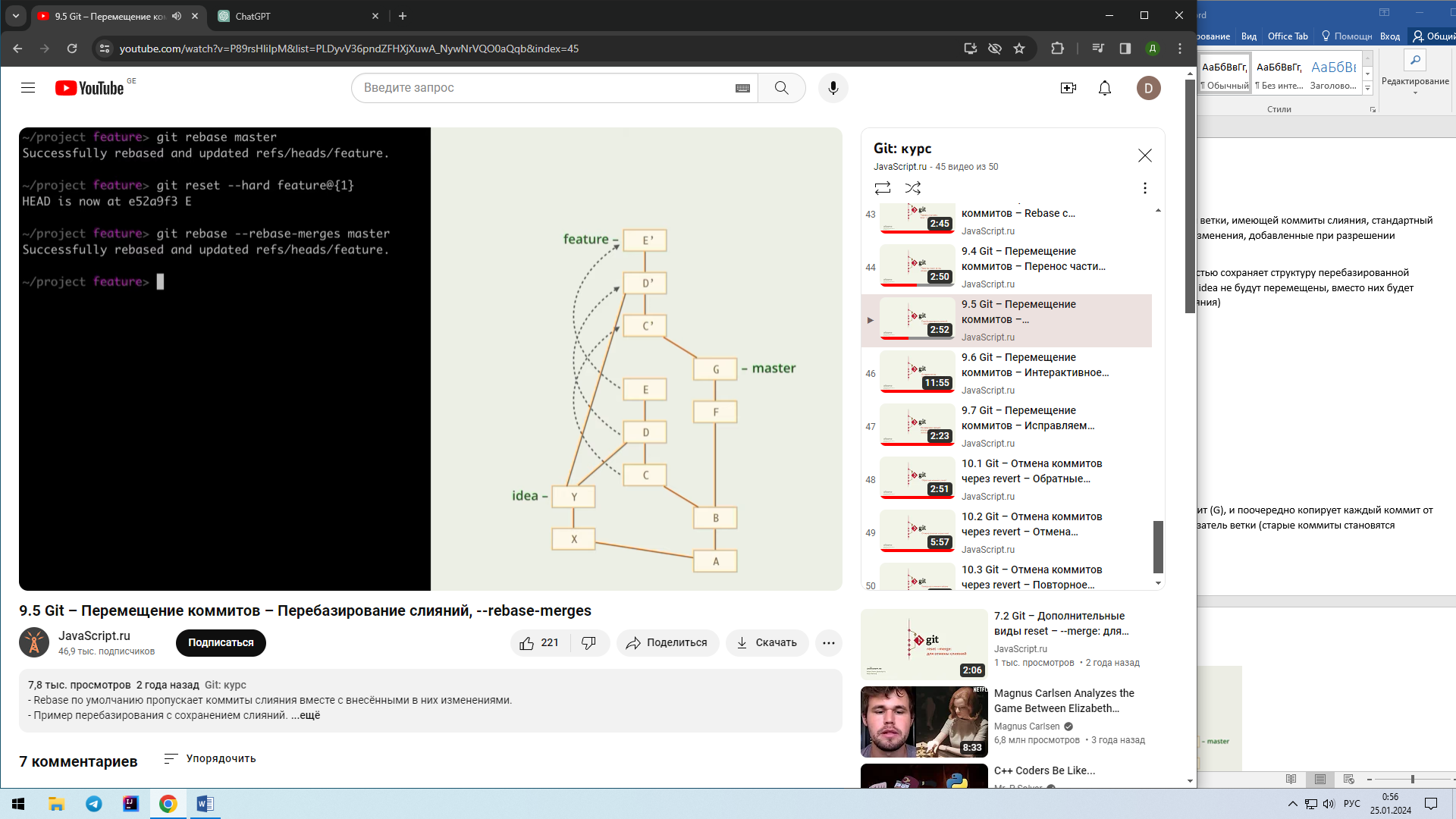
Так же перебазируются только те коммиты, которых нет в master (если бы ветка feature была смержена в master, то ее коммиты бы не включились в rebase)

Если необходимо перенести только ветку fix, то нужно использовать флаг --onto master feature (На ветку мастер начиная с указателя feature)

*Перебазирование при наличии слияния*

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)



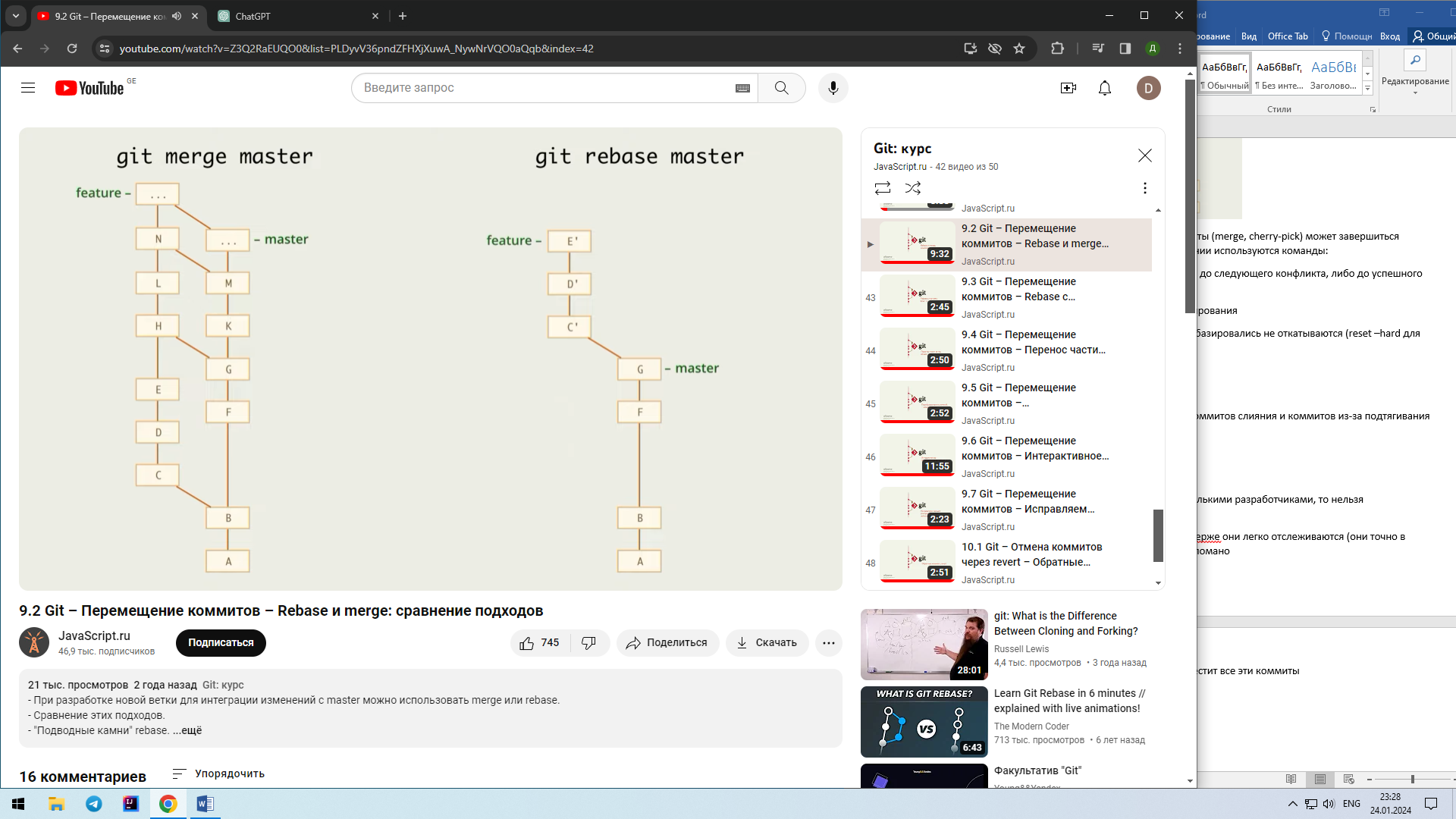
–x “файл теста” – запускает тест при каждом успешном перебазировании коммита, rebase остановится, если тест упадет.

rebase –I – интерактивное перебазирование, то есть для каждого копируемого коммита будет предложено выбрать определенное действие (перебазирование, пропуск, изменение комментария, объединение и тд.). При интерактивном перебазировании с флагом –autosquash, все коммиты, помеченные флагом fixup, будут объединены в один коммит с тем, который они исправляли - может быть нужно, для редактирования коммитов, которые находятся в середине ветки и флаг –amend не применить.

Для разрешения повторного конфликта используется механизм rerere, который автоматически разрешает уже решенные конфликты (применяет решения конфликтов, которые были приняты ранее в такой же ситуации, в тех же файлах) Включение - git config rerere.enabled true (если был включен до конфликта, то нужно запустить скрипт rerere-train)

**Merge или rebase:**

Rebase – Упрощает чтение истории разработки (за счет отсутствия коммитов слияния и коммитов из-за подтягивания изменений из master)



Минусы:

Нельзя перебазировать публичные ветки из-за того, что придется сначала решить все конфликты, а потом создать еще один мерж коммит ([подробнее](https://habr.com/ru/companies/otus/articles/352640/))

Так же могут быть семантические конфликты, которые при мерже легко отслеживаются (они точно в коммите слияния), а при rebase непонятно, где именно все сломано.

\*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 мержа](https://habr.com/ru/articles/195674/), Бранч-стратегии 1, 2, [pull request](https://habr.com/ru/articles/125999/),