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

***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 |
| 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 |
| 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 и сразу выполняет add для этого изменения  [Механика работы команды.](#rm) | -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 + команды для веток** | | | |
| 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 |  |
| merge | Слияние веток |  | Указатель Head на той ветке, в которую скидываются изменения |
| rebase |  |  |  |
| **\*\*\*** | | | |
| ^ | Знак исключения |  |  |
| / | поиск |  |  |
| Gui | Вызов графического интерфейса Gui | gui& | Вызовется в новом окне |
| ORIG\_HEAD | там лежит предыдущая ссылка 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  add <имя> <Ссылка на репозиторий>  remove<Имя>  rename <Старое имя><Новое имя> | Показывает список всех связанных с компьютером репозиториев  Связывает новый удаленный репозиторий. Имя – задаем имя этой ссылке, обычно origin  Удаление удаленного репозитория  Переименование удаленного репозитория |
| push | Отправляет изменения на связанный удаленный сервер | <имя удаленного сервера><ветка>  -u –связывает ветку в локальном и удаленном репозиториях, теперь можно просы вызывать git pull и git push без пути | При выполнении команды push, в ситуации когда ветки слежения удаленного и локального репозитория не совпадают, git попроси выполнить pull (под капотом вызывается fetch и merge) [Разбор](#push) |
| pull | Подтягивает изменения с сервера на комп | <имя удаленного сервера><ветка> |
| fetch | Обновления состояния веток слежения (подгрузит недостающие коммиты в случае несовпадения) |  |  |

[\*Теория и визуализация команд\*](https://www.youtube.com/playlist?list=PLDyvV36pndZFHXjXuwA_NywNrVQO0aQqb)

Ветка – изолированный поток разработки, коммиты каждой ветки будут видны только в текущей.

Подход к разработке – Тематические ветки, релизные ветки, git flow

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

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.

**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).

\_\_\_\_\_\_\_\_\_\_\_

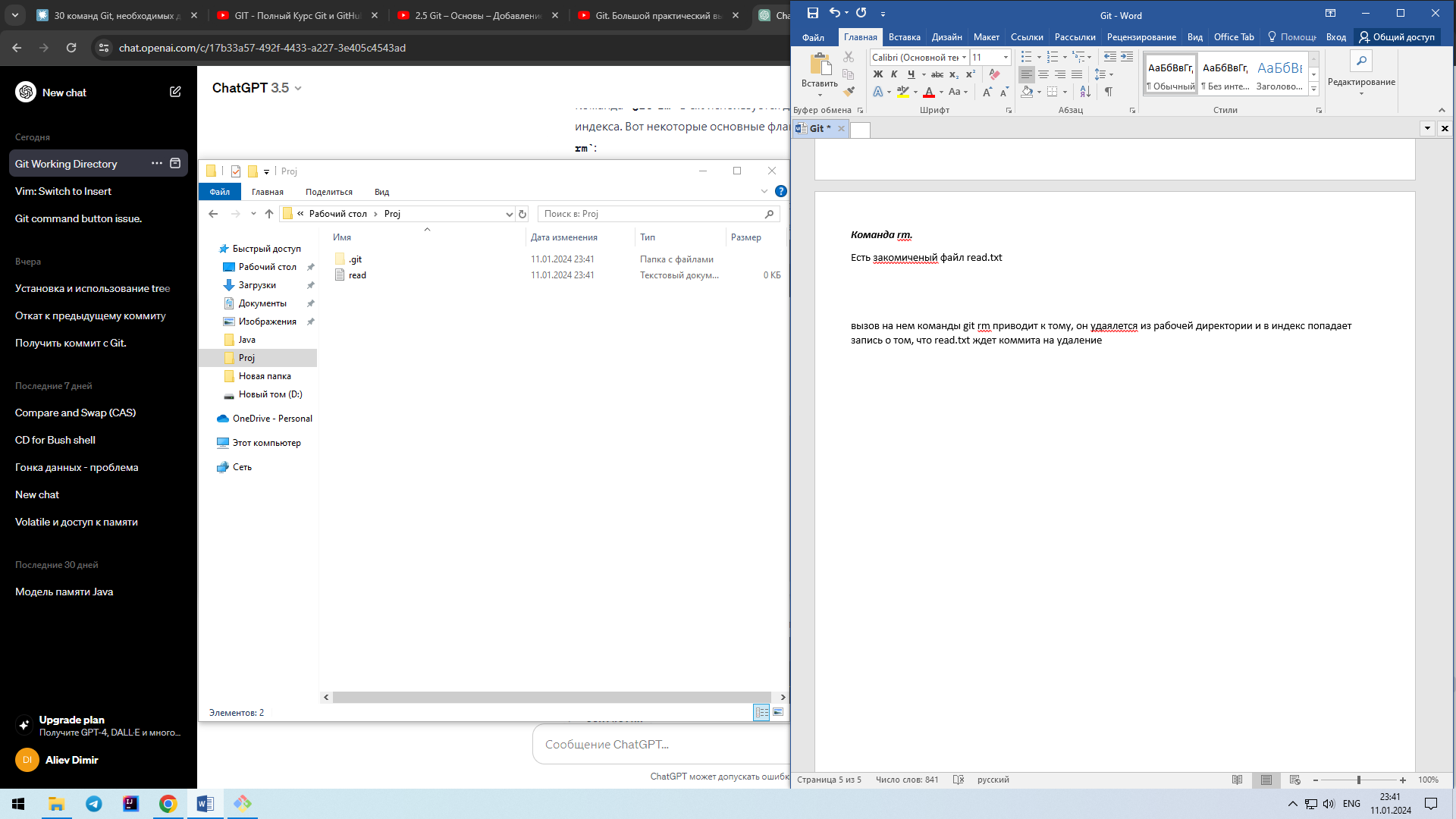
**Виды merge:**

|  |  |  |
| --- | --- | --- |
| Fast Forward (Перемотка): | Во время слияния не возникает конфликтов, перемещает указатель ветки вперед |  |
| True merge (Истинное слияние) | Создание коммита слияния (имеет двух родителей), ссылается на 2 других коммита  Действует по стратегии resolve | Сливаемые ветки не являются прямыми потомками друг друга, поэтому указатель ветки нельзя просто передвинуть вперед |
| При возникновении конфликта называется прерванным слиянием – незаконченный merge можно отменить reset –hard – но затрет все незакоммиченные изменения | Merge –abort  Merge --continue = git commit  Reset –merge |  |
| Стратегии слияния???  Семантические конфликты –no-commit для того, чтобы остаться в состоянии конфликта, даже если слияние успешное (не создает коммит)  --no-ff – вместо fast forward делает коммит слияния (из-за того, что при ff непонятно, где было начало ветки, которую слили, а где конец) | Resolve – находится общий предок для веток и коммиты сливаемых веток. За основу берется общий предок и сравниваются изменения с коммитами веток.  Ситуации:  1) Файл не изменился ни в одной ветке  2) Файл изменился в одной из веток – берется версия измененного  3)Файл изменился в обеих ветках – конфликт (прерванное слияние), надо решать руками | При конфликтах можно переключаться между версиями файлов git checkout –ours/--theirs <Имя файла> |
|  |  |  |

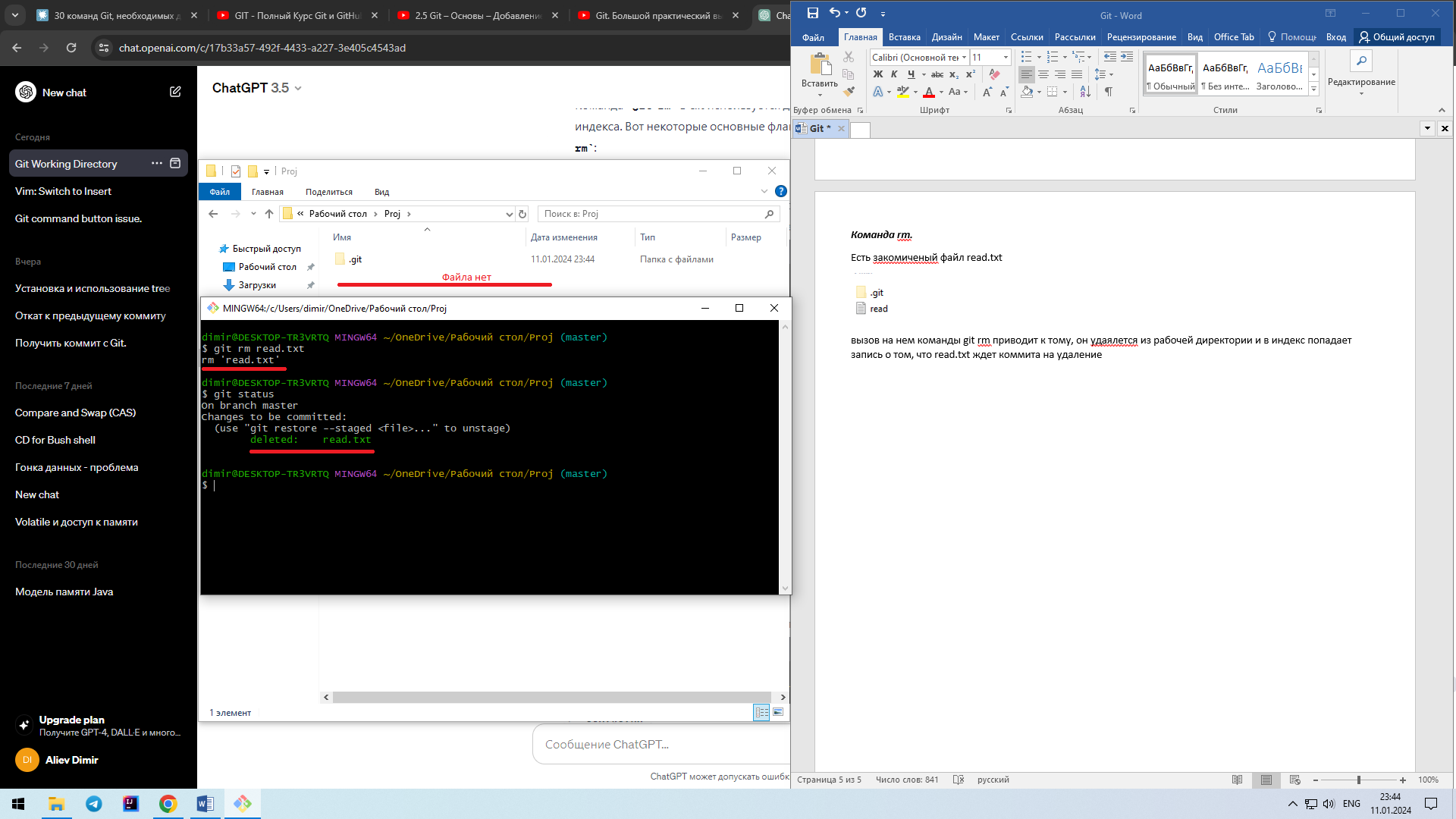
\_\_\_\_\_\_\_\_\_\_\_\_

***Команда rm***

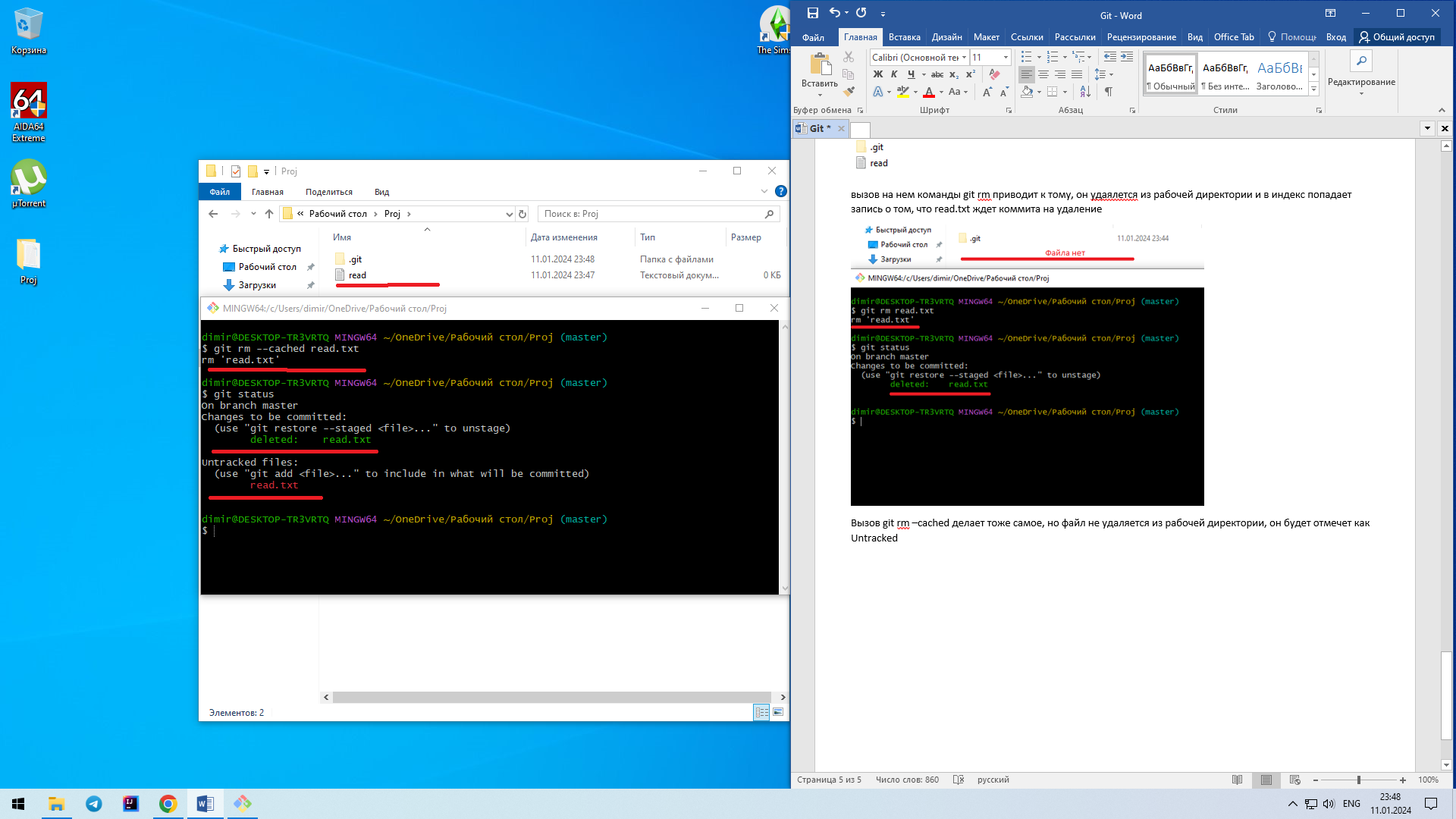
Есть закомиченый файл read.txt



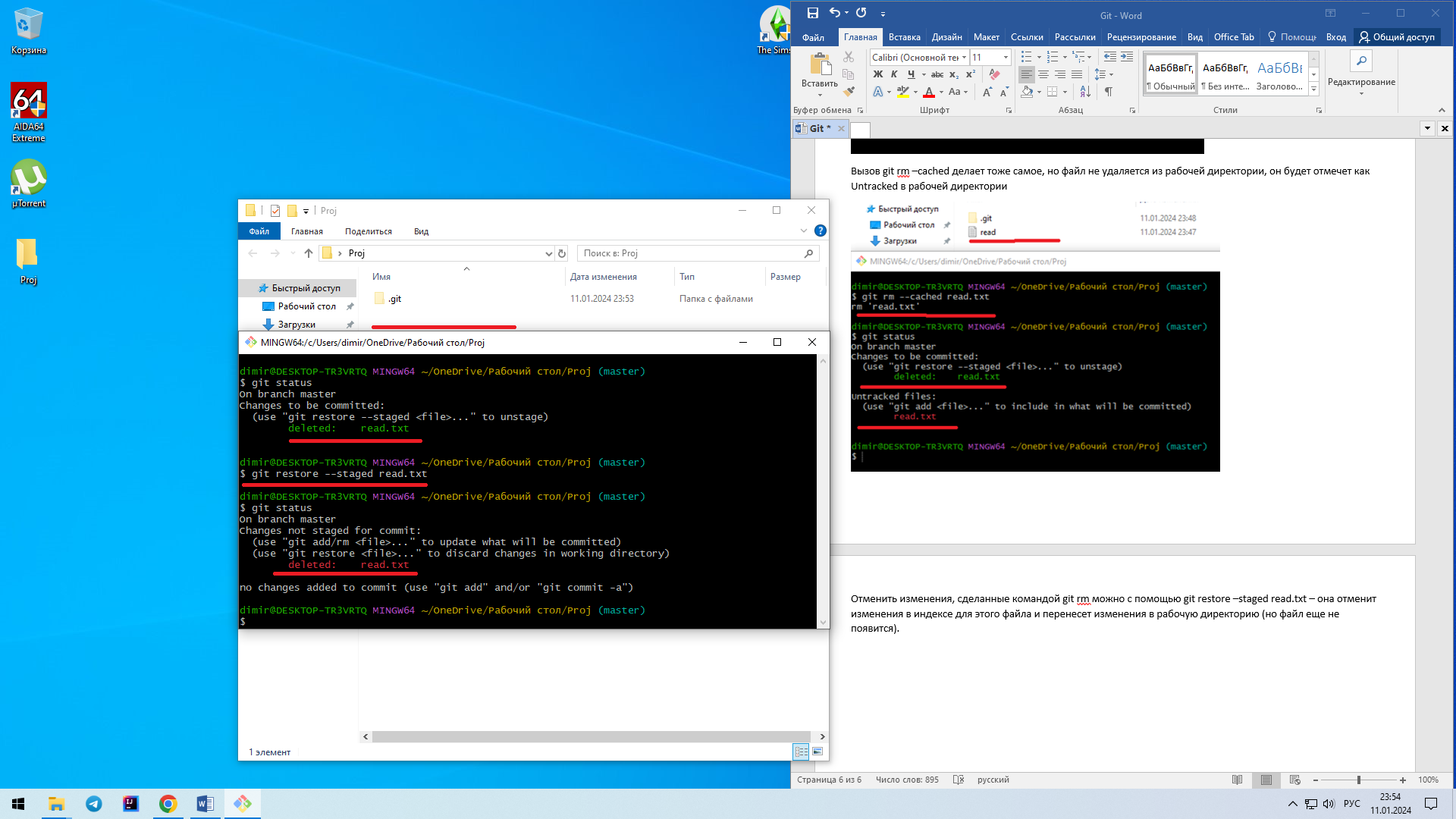
Вызов на нем команды git rm приводит к тому, он удаляется из рабочей директории и в индекс попадает запись о том, что read.txt ждет коммита на удаление



Вызов git rm –cached делает тоже самое, но файл не удаляется из рабочей директории, он будет отмечет как Untracked в рабочей директории

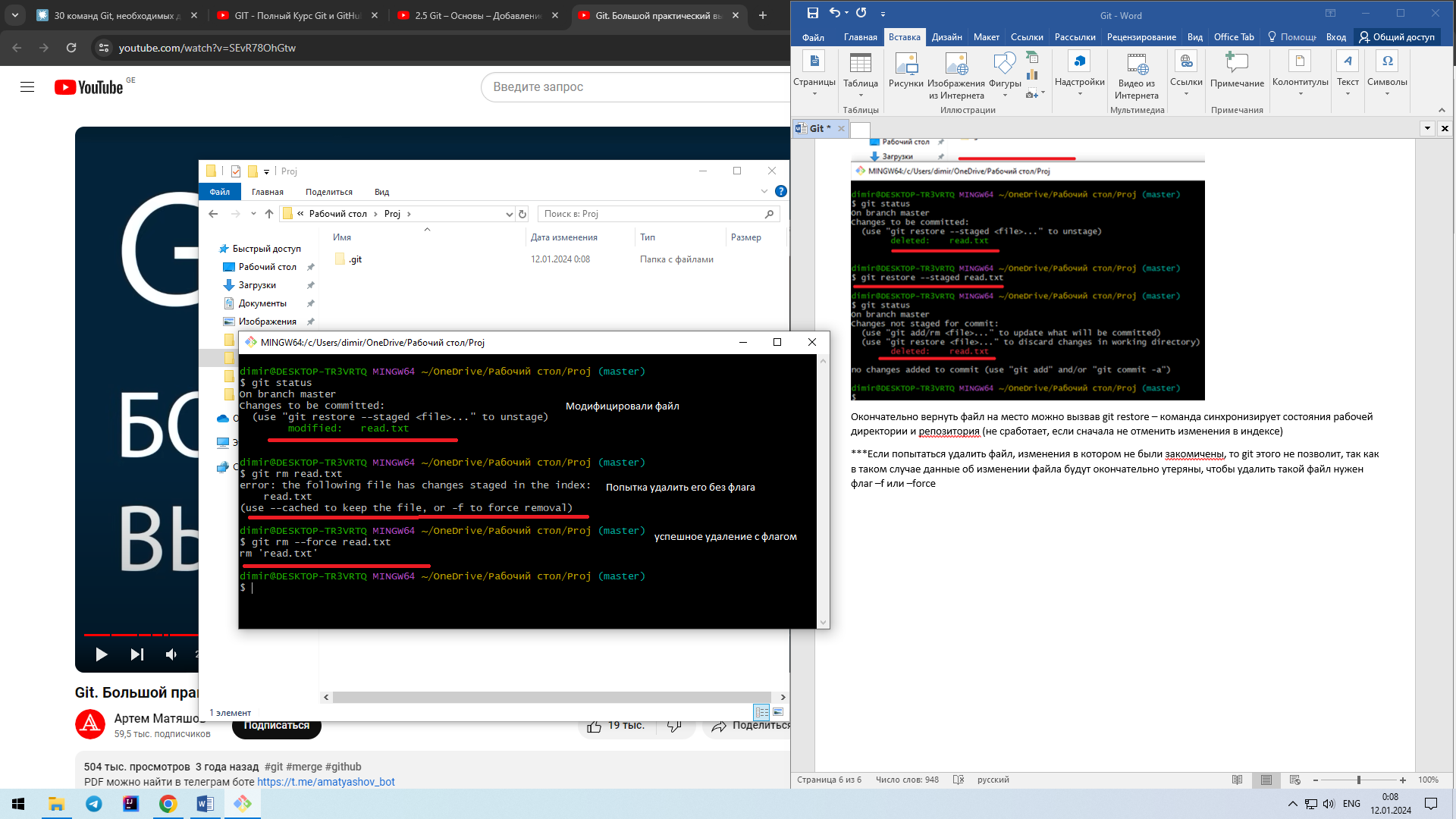


Отменить изменения, сделанные командой git rm можно с помощью git restore –staged read.txt – она отменит изменения в индексе для этого файла и перенесет изменения в рабочую директорию (но файл еще не появится).



Окончательно вернуть файл на место можно вызвав git restore – команда синхронизирует состояния рабочей директории и репозитория (не сработает, если сначала не отменить изменения в индексе)

\*\*\*Если попытаться удалить файл, изменения в котором не были закомичены, то git этого не позволит, так как в таком случае данные об изменении файла будут окончательно утеряны, чтобы удалить такой файл нужен флаг –f или –force



***Команда reset***

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

--keep

***git push при наличии правок***

При связывании локального и удаленного репозитория создается ветка слежения, нужен для понимания того, на каком коммите находится указатель в удаленном репозитории. Этот указатель нельзя перемещать вручную, двигается только при команде pull/push

Git fetch – если кто-то другой сделал push в ветку, все эти коммиты подгрузятся в отдельное ответвление и передвинет ветки слежения.

Push – если сервер, принимая правки может просто передвинуть ветку слежения вперед, то он примет их, если же требуется какой-либо merge, то сервер отклонит эти правки

При подгрузке проекта он отображает только ветку мастер, вроде как, чтобы подгрузить другую ветку надо переключиться на нее checkout и она подгрузится, то есть, если подгрузить проект, с несколькими ветками, потом вызвать branch, видно будет только ветку master

Rebase – переписывает историю ветки

Git flow – вроде, организация работы с ветками, когда ты перед коммитом сначала подгружает новые изменения из родительской ветки и тестируешь свой код с актуальными изменениями перед мержом.

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.

Cherry-pick команда, применимая в ситуации, когда из одной ветки в другую необходимо перенести один или несколько коммитов, а не всю историю (не используя merge или rebase), то есть из вашей ветки в мастер перенесется 1 коммит, а не вся история