**Практическая работа №3**

**Построение архитектуры программного средства**

**Цель работы:** приобрести навыки разработки архитектуры программного средства к проекту

**Порядок выполнения работы**

1. Ознакомиться с краткими теоретическими сведениями и заданиями к работе.
2. Выполнить задания работы.
3. Оформить отчет о выполнении.
4. Сделать выводы.
5. Ответить на контрольные вопросы.

**Используемые источники:**

<https://medium.com/@breadcrumbszone/%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D1%82%D1%8C-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%B9-github-71ebe3ace9b2>

<https://webformyself.com/urok-1-git-i-github-vvedenie-chto-takoe-git-i-github/>

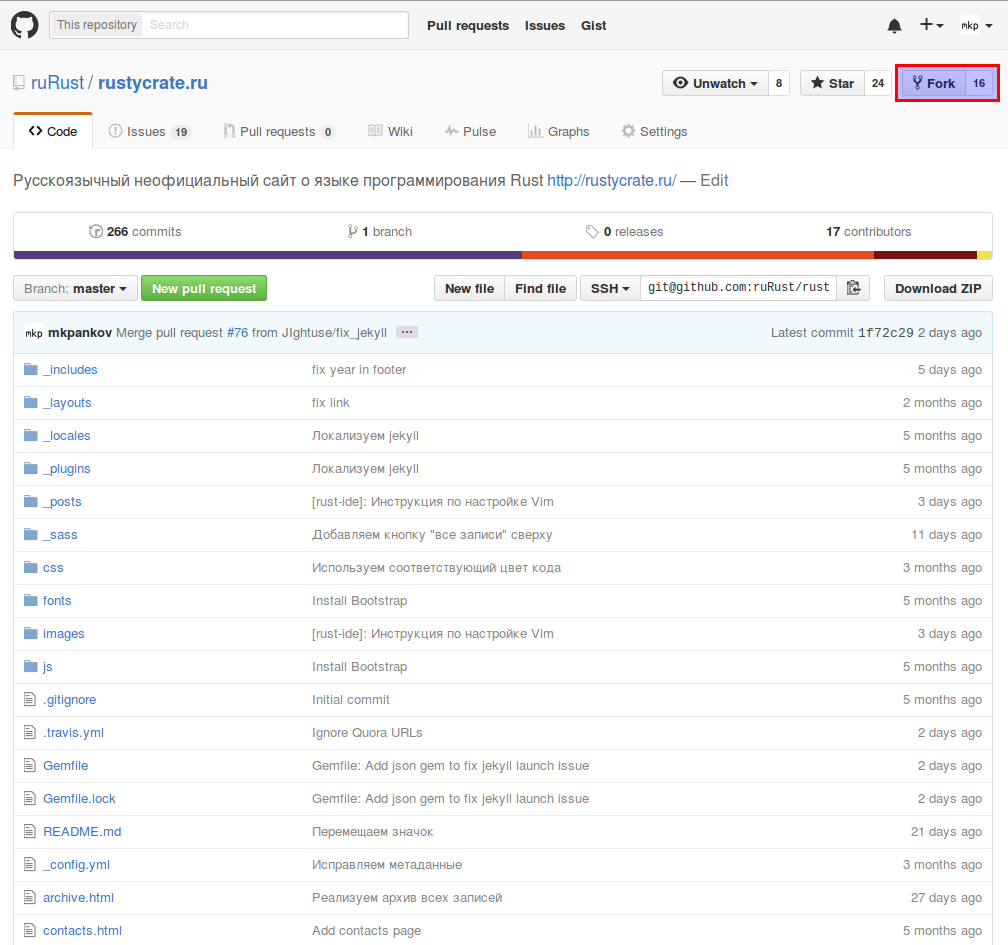
**МЕТОДИЧЕСКИЕ УКАЗАНИЯ**

[**https://rustycrate.ru/%D1%80%D1%83%D0%BA%D0%BE%D0%B2%D0%BE%D0%B4%D1%81%D1%82%D0%B2%D0%B0/2016/03/07/contributing.html**](https://rustycrate.ru/%D1%80%D1%83%D0%BA%D0%BE%D0%B2%D0%BE%D0%B4%D1%81%D1%82%D0%B2%D0%B0/2016/03/07/contributing.html)

## Форкаем проект

Вы не можете отправлять коммиты (git push) напрямую в исходный репозиторий. По желанию хозяин проекта может это разрешить, но обычно доступ на запись есть только у людей, поддерживающих проект, а все остальные работают через Pull Request’ы («запросы на вливание изменений»; о них — ниже).

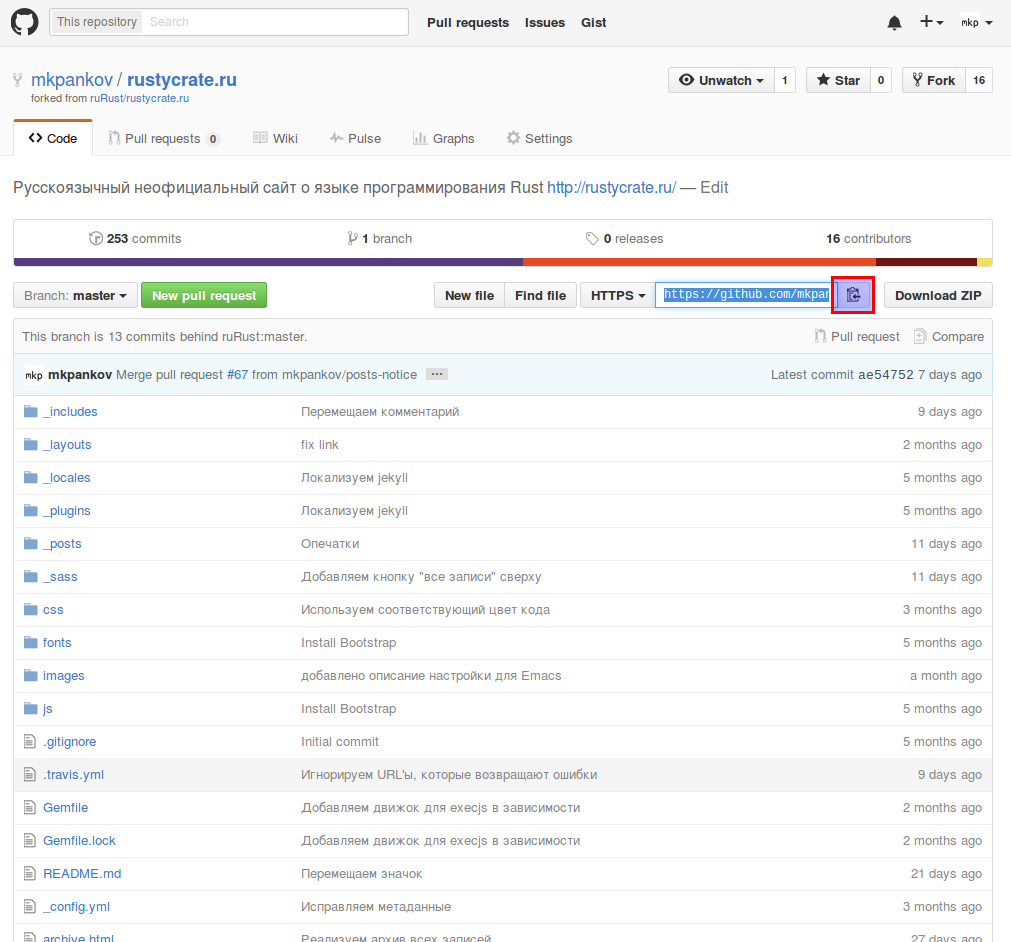
Поэтому мы форкаем проект — это создаст копию репозитория в вашем аккаунте. При этом у вас появится доступ на запись в вашу копию.



Через мгновение вы будете перенаправлены на страницу вашего форка.

## Клонируем репозиторий

Затем нужно [склонировать репозиторий](https://git-scm.com/book/ru/v1/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-Git-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F" \l "%D0%9A%D0%BB%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D1%81%D1%83%D1%89%D0%B5%D1%81%D1%82%D0%B2%D1%83%D1%8E%D1%89%D0%B5%D0%B3%D0%BE-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F) на вашу локальную машину. Для этого нам нужен URL репозитория. Нажав на кнопку справа, вы скопируете его в буфер обмена. Обратите внимание на выбранный слева протокол. Если вы не настраивали SSH для GitHub, там должно быть указано HTTPS.



Затем выполняем команду в терминале (или командной строке Windows):

git clone <вставляем\_URL>

Репозиторий склонируется в под-директорию текущей директории. Например, если репозиторий называется foobar, у вас появится каталог foobar.

## Создаём ветку

[Ветка](https://git-scm.com/book/ru/v1/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-Git-%D0%A7%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-%D0%B2%D0%B5%D1%82%D0%BA%D0%B0%3F) по умолчанию — master. Чтобы изменениями было проще управлять и они не смешивались друг с другом, создадим отдельную ветку, где и будем работать. При этом ветку стоит назвать так, чтобы имя говорило о её назначении.

Например, мы хотим исправить ошибку в реализации протокола PROTOBAZ, так что наша ветка будет называться fix-protobaz.

Теперь заходим в наш склонированный репозиторий и [создаём ветку](https://git-scm.com/book/ru/v1/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-Git-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-%D0%B2%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F-%D0%B8-%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D1%8F):

|  |  |
| --- | --- |
| 1  2 | cd foobar  git checkout -b fix-protobaz |

Вторая команда создаст ветку и перейдёт на неё (сделает checkout).

Если после этого выполнить git status, он покажет

|  |  |
| --- | --- |
| 1  2 | On branch fix-protobaz  nothing to commit, working directory clean |

Эту команду стоит запомнить — когда не понимаете, в каком состоянии репозиторий, просто выполните её. Чаще всего в её выводе git покажет другие команды, которые делают то, что вы (скорее всего) и хотите сделать.

## Делаем изменения

Теперь приступаем к работе. Редактируем код, обновляем документацию, чиним тесты, дополняем README.

Эти изменения мы коммитим в нашу ветку. Как это сделать — ниже.

При этом старайтесь делать коммиты часто, а сами коммиты — небольшими по объёму. Каждый коммит должен делать ровно одну вещь, и при этом поддерживать работоспособность проекта. Стремиться нужно к тому, чтобы в будущем можно было перейти на любой коммит и получить рабочий проект.

Если у вас сразу не получается придерживаться такой дисциплины, или изменения затрагивают весь проект «насквозь», допустимо ломать проект и постепенно чинить его в следующих коммитах.

Если вы уже достаточно разбираетесь в Git, такие не-атомарные изменения потом нужно объединить в один коммит с помощью [interactive rebase](https://git-scm.com/book/ru/v1/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-Git-%D0%9F%D0%B5%D1%80%D0%B5%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C-%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8" \l "%D0%98%D0%B7%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5-%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D0%B9-%D0%BD%D0%B5%D1%81%D0%BA%D0%BE%D0%BB%D1%8C%D0%BA%D0%B8%D1%85-%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82%D0%BE%D0%B2) и [squash](https://git-scm.com/book/ru/v1/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-Git-%D0%9F%D0%B5%D1%80%D0%B5%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C-%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8" \l "%D0%A3%D0%BF%D0%BB%D0%BE%D1%82%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5-%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82%D0%BE%D0%B2).

Итак, после редактирования файлов мы имеем следующую ситуацию (это вывод git status):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | On branch fix-protobaz  Changes not staged for commit:  (use "git add <file>..." to update what will be committed)  (use "git checkout -- <file>..." to discard changes in working directory)  modified: src/protobaz.rs  no changes added to commit (use "git add" and/or "git commit -a") |

В выводе есть все необходимые вам команды:

* git add <file>... добавляет файл в содержимое коммита, который вы собираетесь записать
* git checkout -- <file>... откатывает ваши изменения файла

Поэтому делаем git add src/protobaz.rs, а затем git commit. Откроется редактор, в котором нужно ввести сообщение коммита.

Сообщение коммита — это описание того, что вы сделали. Его читают другие участники проекта и рецензент. Поэтому оно должно быть осмысленным и читаемым.

Формат сообщения о коммите таков:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Краткое описание коммита (не более 50 символов)  Подробное описание коммита - зачем он сделан, почему нельзя сделать по-другому,  источники информации.  Служебная информация - теги, ссылки на задачи, какие задачи коммит закрывает. |

Например,

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Исправляем длину буфера в реализации PROTOBAZ  Неверное вычисление длины приводило к отсечению части сообщения в ситуации,  когда пользователь использует foobar-1.3.4.  Смотри http://foobar.org/bugs?id=1234  Fix #123 |

Из всех частей сообщения, только первая является обязательной.

Она должна иметь вид <Глагол в настоящем времени, первом лице, множественном числе> <объект изменения>. Говорим о том, что мы делаем в этом коммите: [мы] исправляем ошибку, [мы] добавляем возможность, [мы] обновляем документацию.

Последняя строка — это команда GitHub. Когда коммит с такой командой попадает в master, GitHub автоматически закроет указанную задачу. Можно использовать разные формы этой команды: Fix #123, fixes #123, close #123 и другие. Это экономит время на поддержку проекта.

В наших проектах нужно использовать Fix #123 или Close #123 на последней строке сообщения коммита.

git log --oneline выводит историю в формате «1 коммит — 1 строка на экране». При этом он использует в качестве описания коммита первую строку — краткое описание. Поэтому оно обязательно должно быть отделено пустой строкой от остального описания — иначе однострочный вывод разъедется.

Язык сообщения о коммите должен соответствовать принятому языку проекта. Поскольку наши проекты нацелены на русскоязычную аудиторию и разработчики говорят по-русски, сообщения коммитов также должны быть на русском.

Когда вы ввели сообщение коммита в редакторе, сохранили файл и закрыли его, можно выполнить git log и [убедиться](https://git-scm.com/book/ru/v1/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%9F%D1%80%D0%BE%D1%81%D0%BC%D0%BE%D1%82%D1%80-%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8-%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82%D0%BE%D0%B2), что коммит записан в историю.

## Проверяем изменения

Когда вы сделали правки, стоит их проверить — если только это не что-то абсолютно тривиальное.

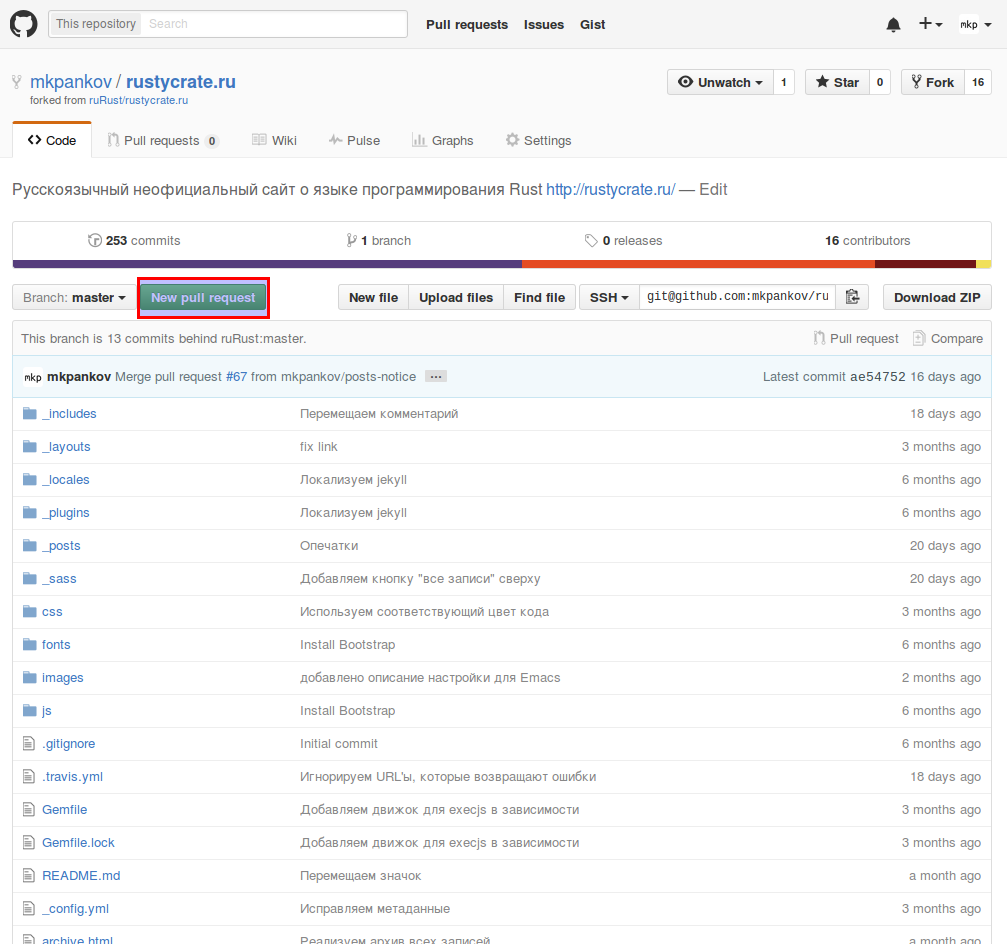
Для этого нужно собрать проект и запустить тесты, если они есть. В любом случае стоит проверить работу кода, который вы написали или изменили, запустив программу или вызвав библиотеку.

Если проект — это статически генерируемый сайт, то сгенерируйте его локально и убедитесь, что ничего не отвалилось и вёрстка не разъехалась. Если книга — то же самое. Смотрите по крайней мере на те места, которые вы правили.

## Создаём Pull Request

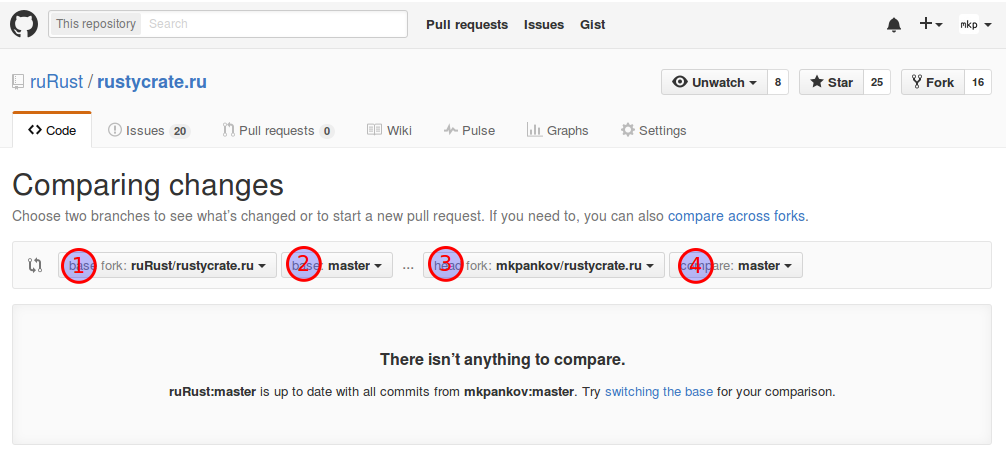
Когда работа и проверка закончены, пора создавать Pull Request. Pull Request — это запрос на вливание изменений из вашей ветки в основную ветку исходного репозитория. Таким образом они попадут к хозяевам проекта.

Чтобы создать Pull Request, зайдём на страницу вашего форка. Справа от выпадающего меню с выбором ветки есть кнопка «New pull request».



Нажимаем её.

Вы попадаете в окно сравнения веток.

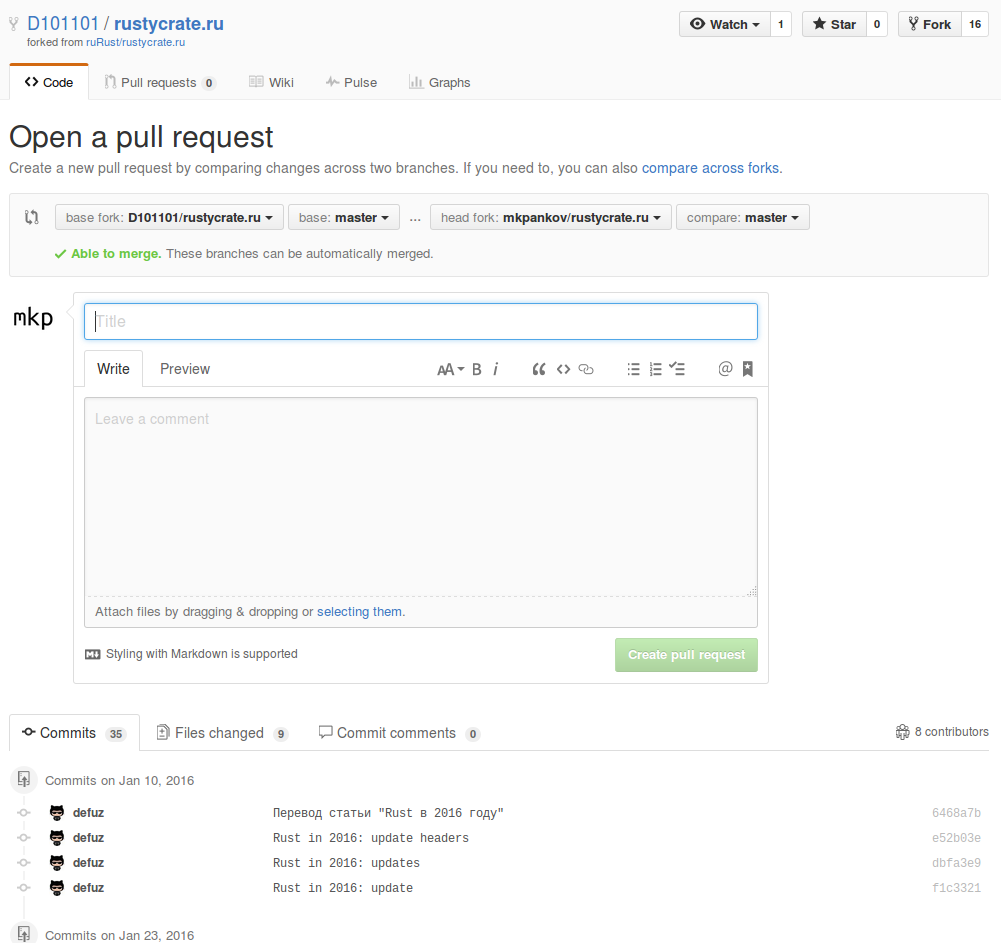


Вот элементы этого окна, по порядку:

1. Базовый репозиторий, в который будет создаваться PR. Это должен быть репозиторий, от которого вы делали форк. Если вы форкнули проект ruRust/rustycrate.ru, а ваше имя пользователя GitHub — user, то у вас будет проект user/rustycrate.ru.
2. Базовая ветка в этом репозитории, обычно master.
3. Репозиторий, откуда должны вливаться изменения. Здесь должен быть выбран репозиторий в вашем аккаунте — user/rustycrate.ru.
4. Ветка, откуда будут вливаться изменения. Это должна быть ветка, которую мы создали в разделе «Создаём ветку».

Дальше просмотрите изменения — то ли это, что вы делали? Если да, то нажимайте кнопку «Create pull request». В моём примере её нет, т. к. ветки в форке и в оригинале находятся в одинаковом состоянии. В вашем же случае внизу будет список коммитов, которые попадут в исходный репозиторий, и, на других вкладках — сами изменения и комментарии к изменениям.

После нажатия кнопки появится окно ввода сообщения Pull Request.



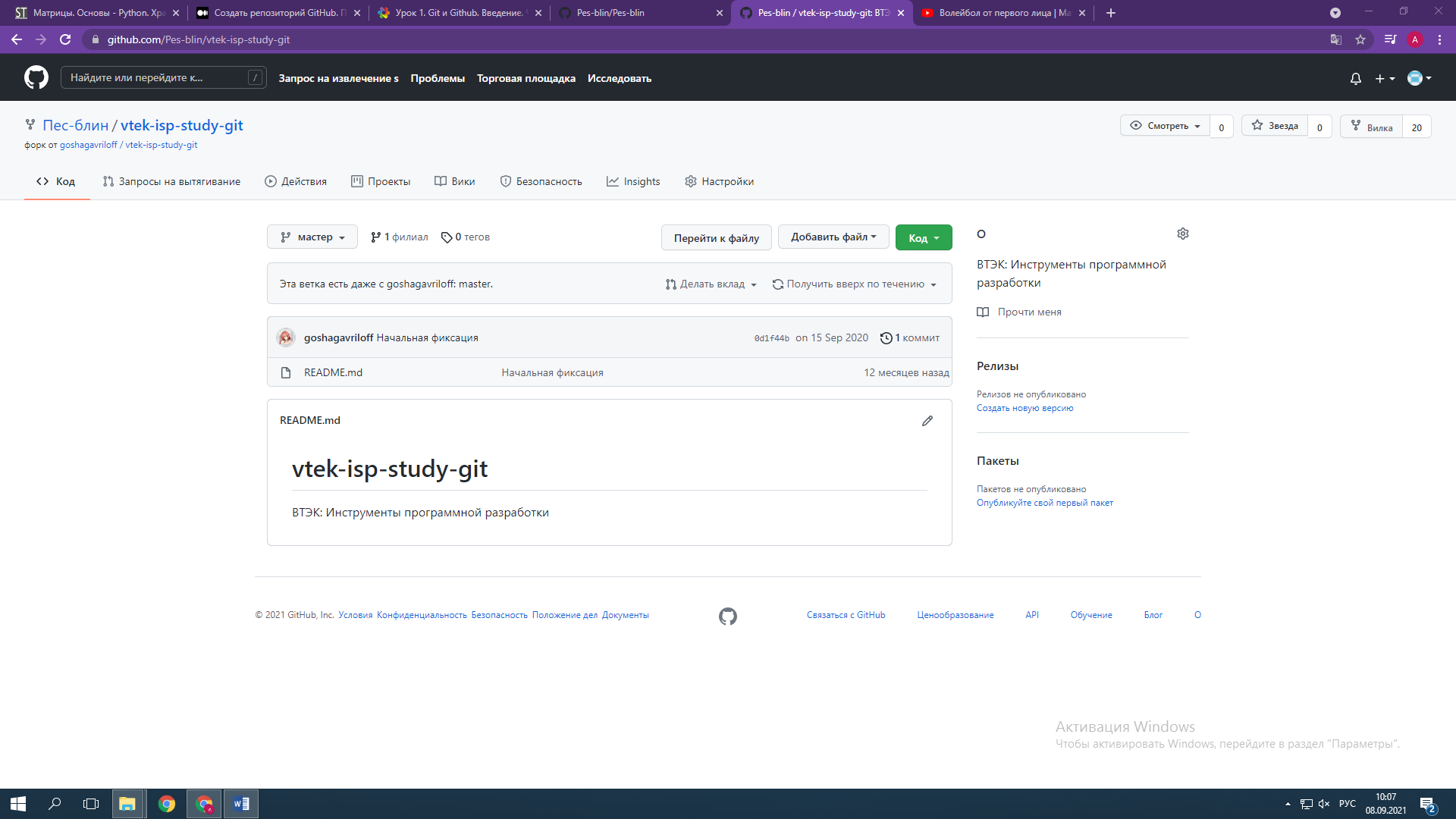
Сообщение PR — это описание того, что сделано и зачем. В отличие от сообщения коммита, здесь уже нужно писать высокоуровневое описание того, какие изменения сделаны.

[**https://rustycrate.ru/%D1%80%D1%83%D0%BA%D0%BE%D0%B2%D0%BE%D0%B4%D1%81%D1%82%D0%B2%D0%B0/2016/03/07/contributing.html**](https://rustycrate.ru/%D1%80%D1%83%D0%BA%D0%BE%D0%B2%D0%BE%D0%B4%D1%81%D1%82%D0%B2%D0%B0/2016/03/07/contributing.html)

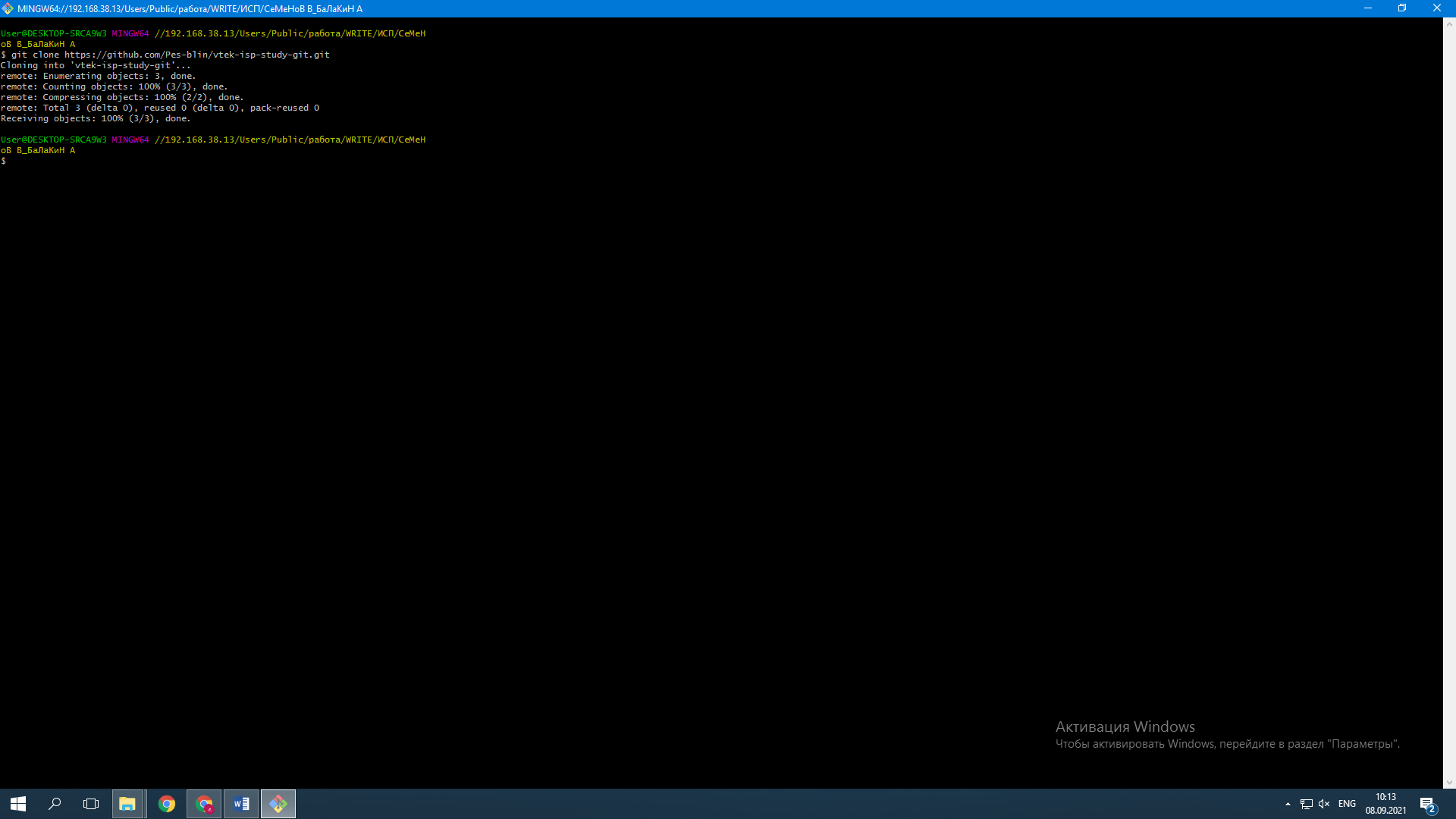
**ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ**

Вам потребуется аккаунт на GitHub. Регистрация бесплатная и требует указания лишь имени пользователя и электронной почты.

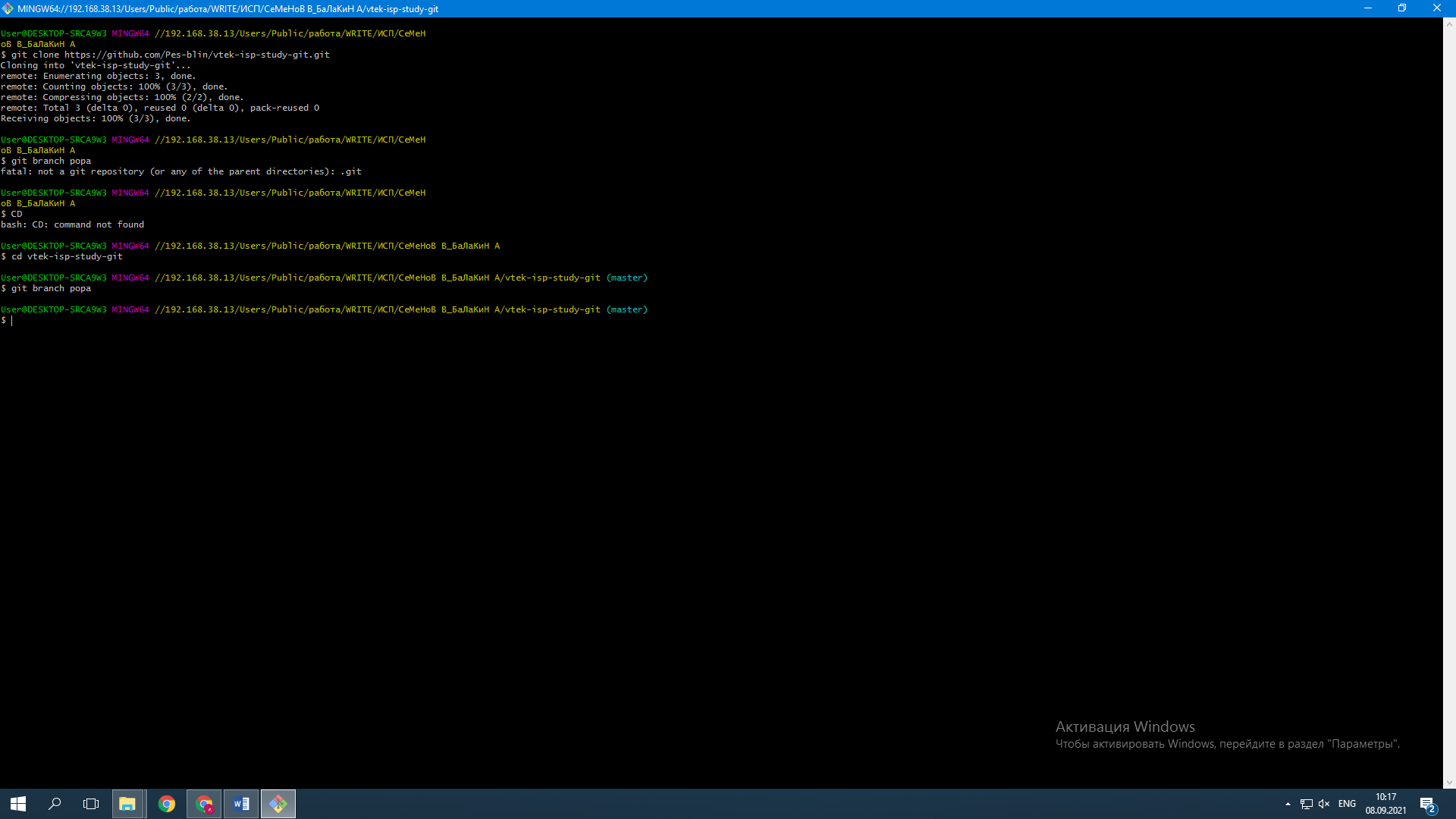
1. Форкните проект <https://github.com/goshagavriloff/vtek-isp-study-git>.



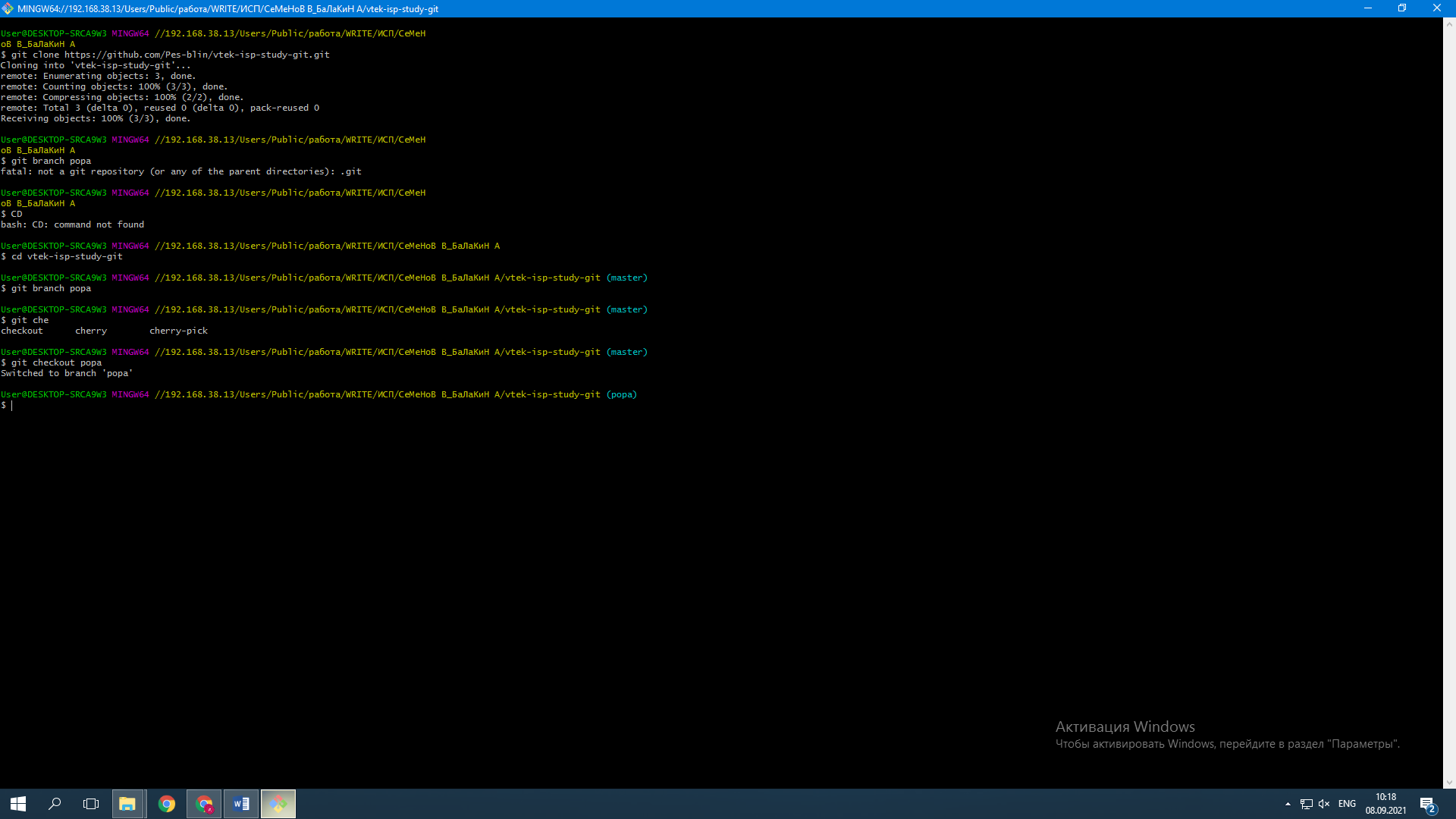
1. Склонируйте репозиторий.



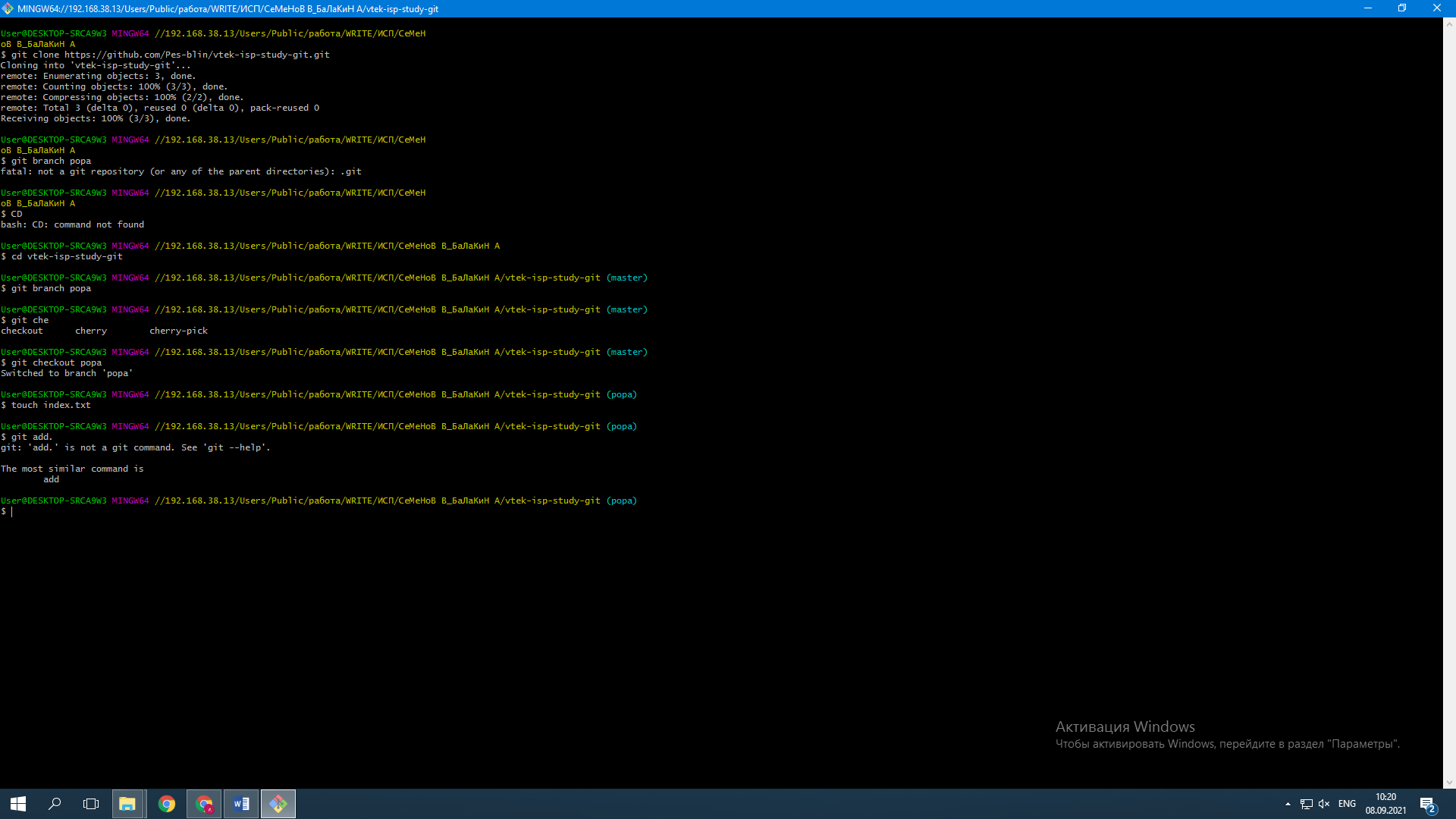
1. Создайте ветку для своей работы.



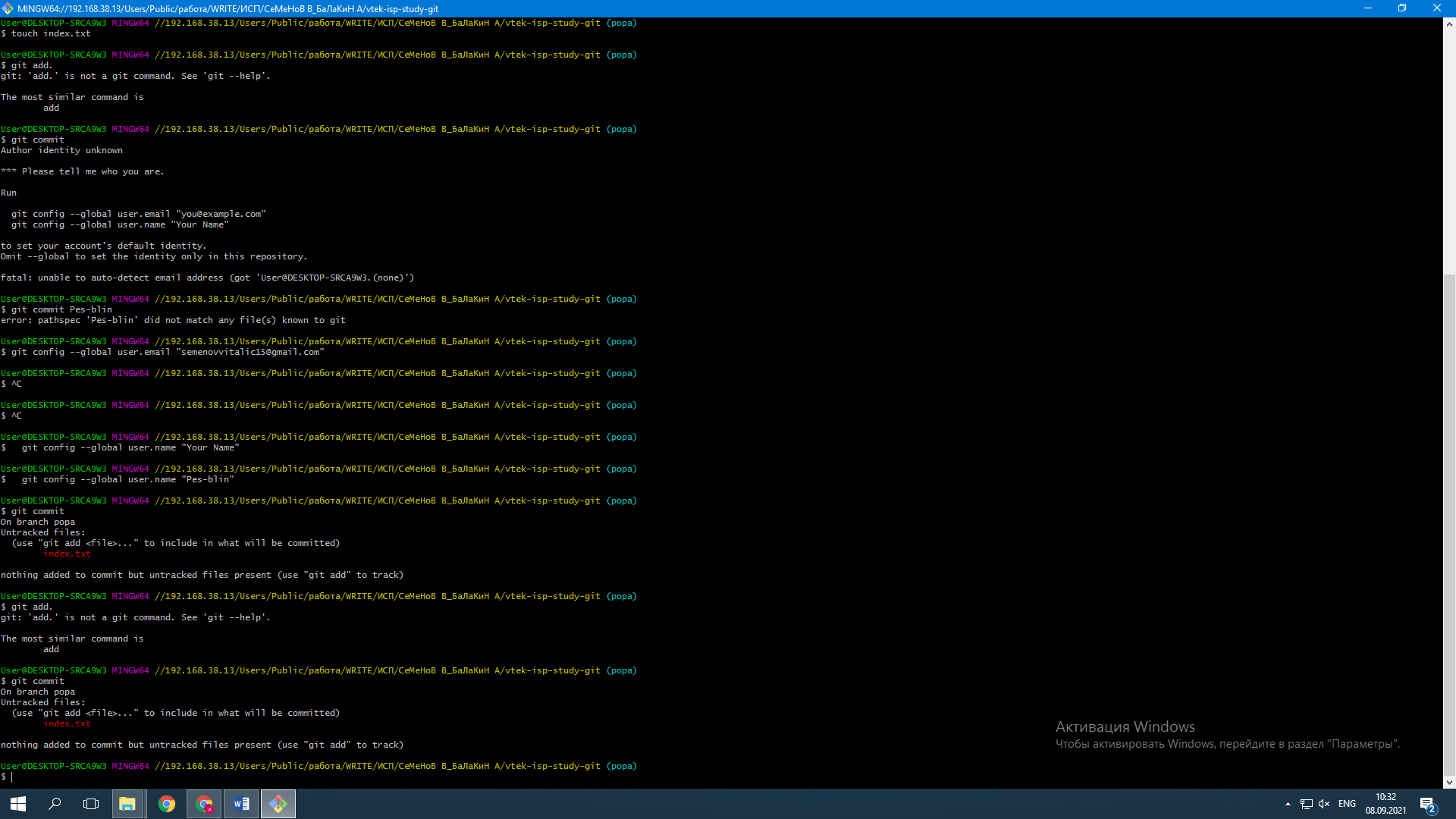
1. Выполнять синхронизацию рабочей папки с репозиторием.



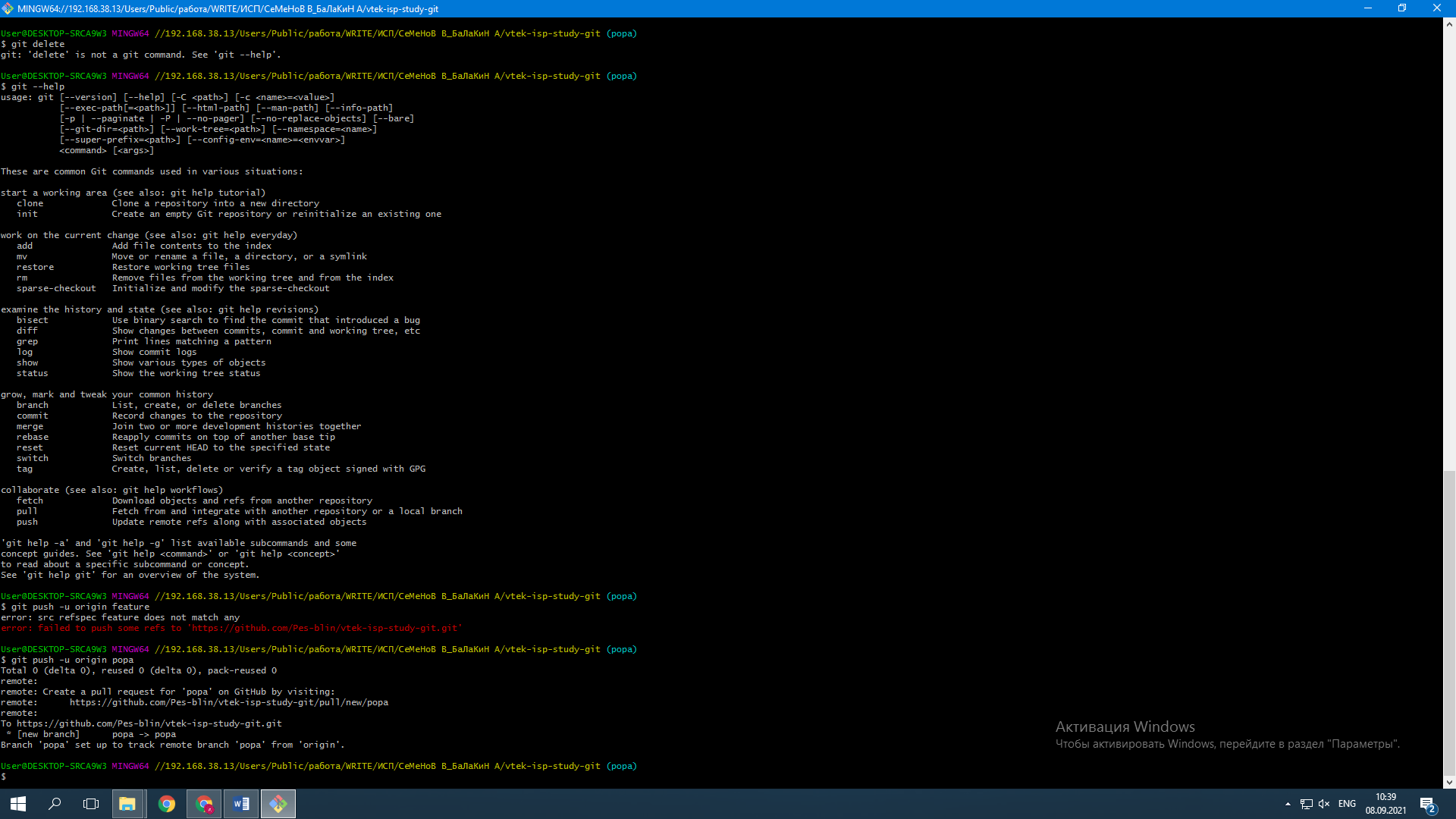
1. Добавить файлы в репозиторий



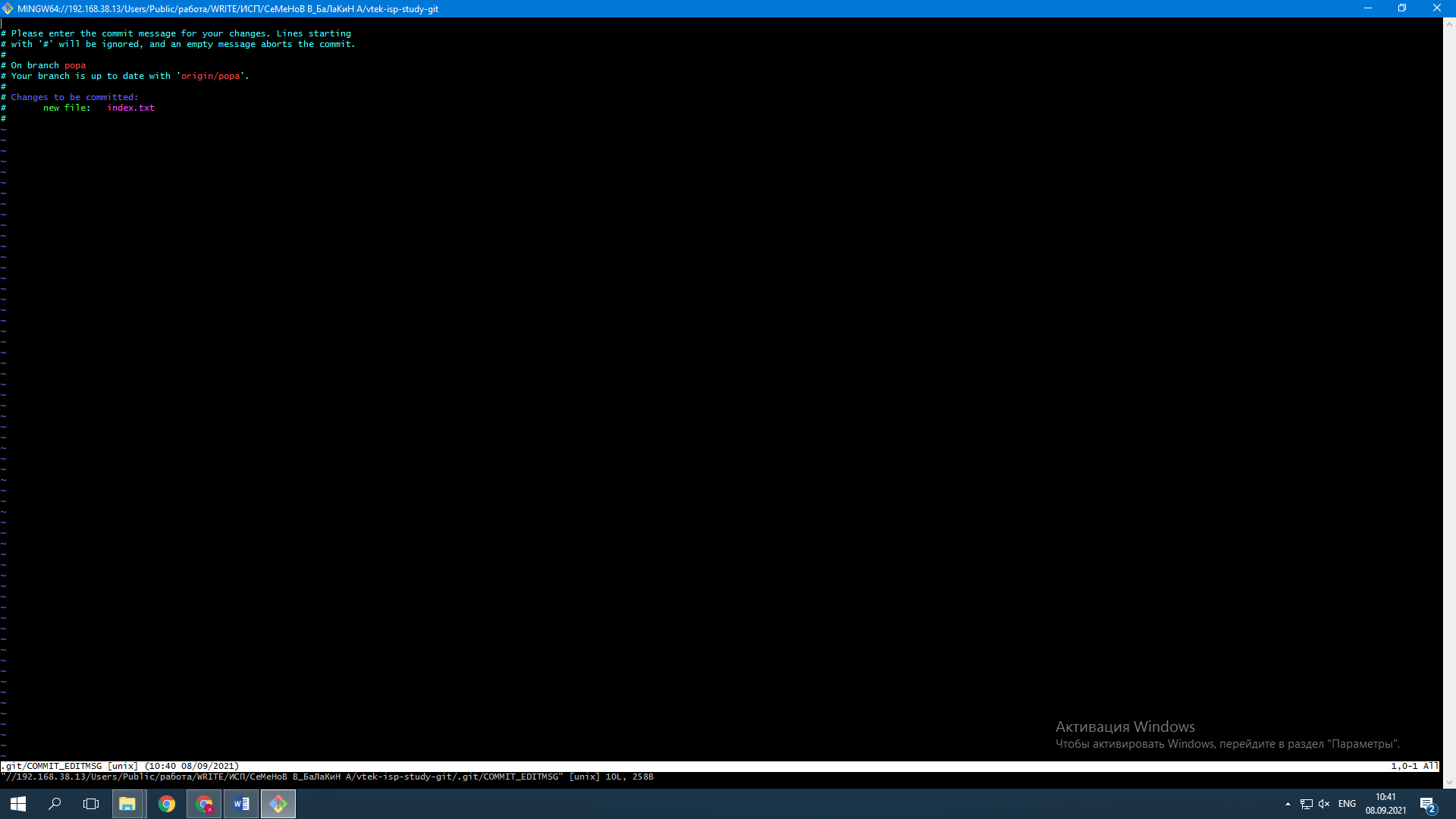
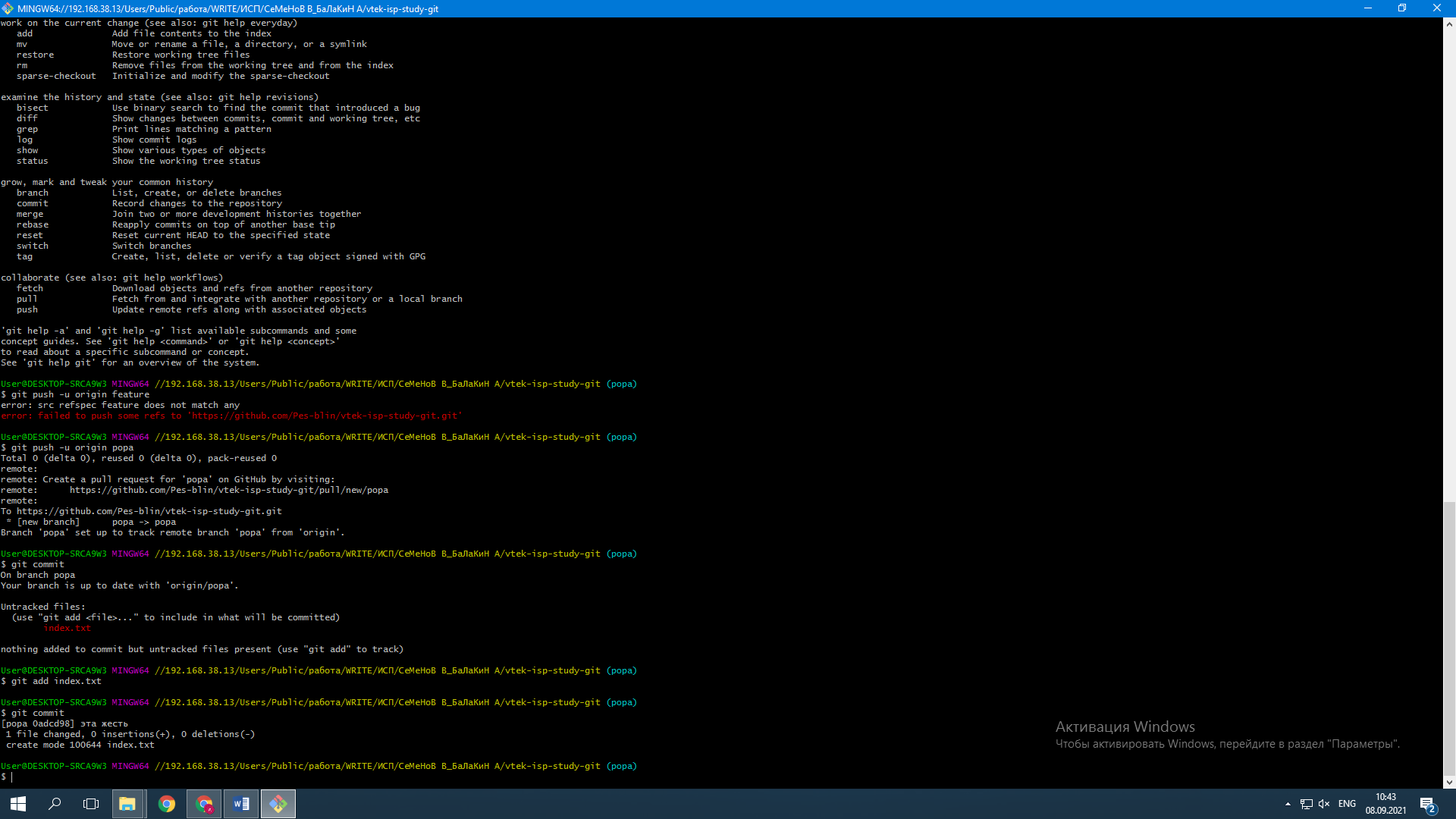
1. Зафиксируйте изменения(commit).



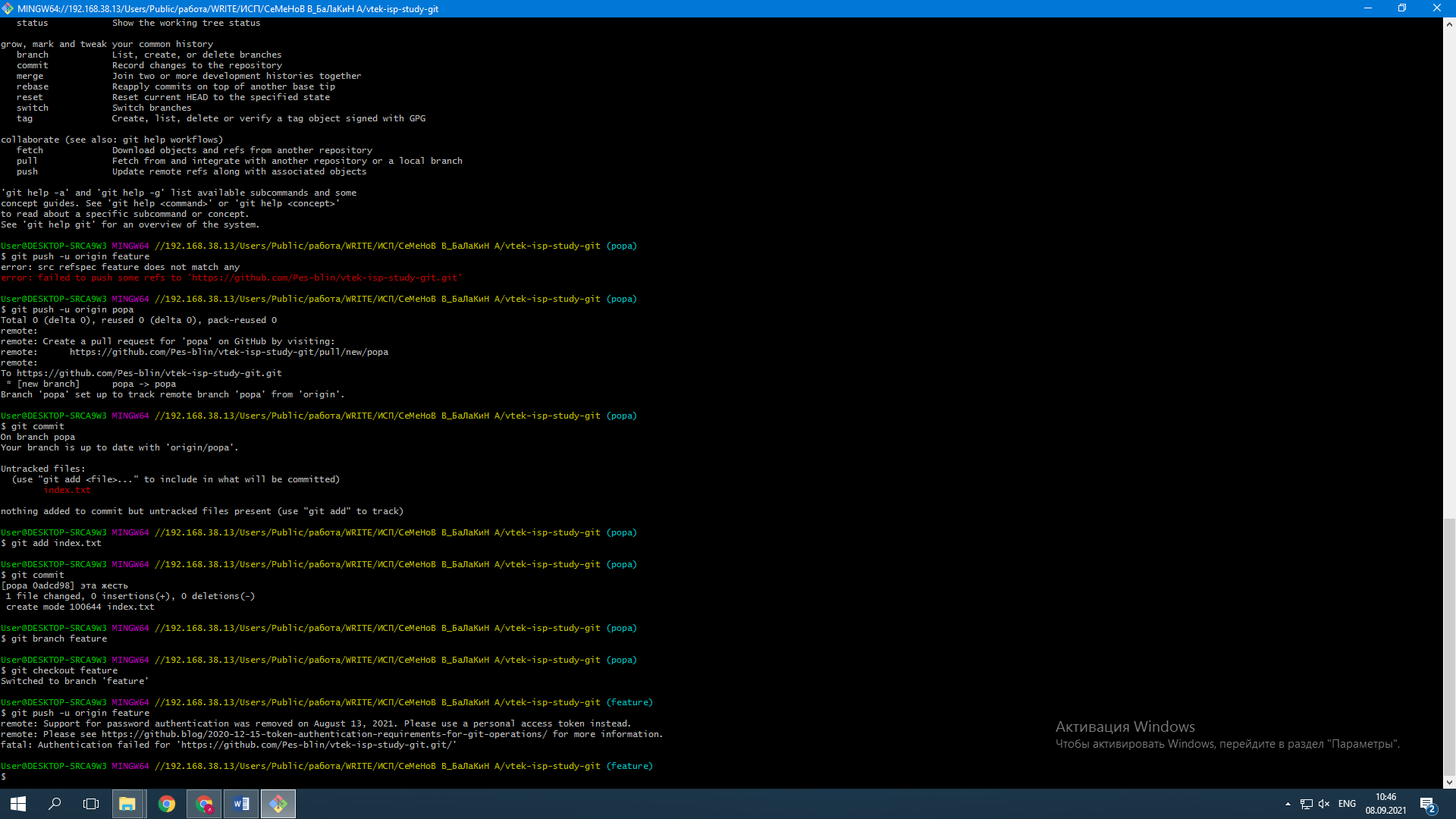
1. Удалить файл из репозитория

ghp\_DHyDKb0pP7CSvF60wdg24kJw2KtXSR2i7aHE 

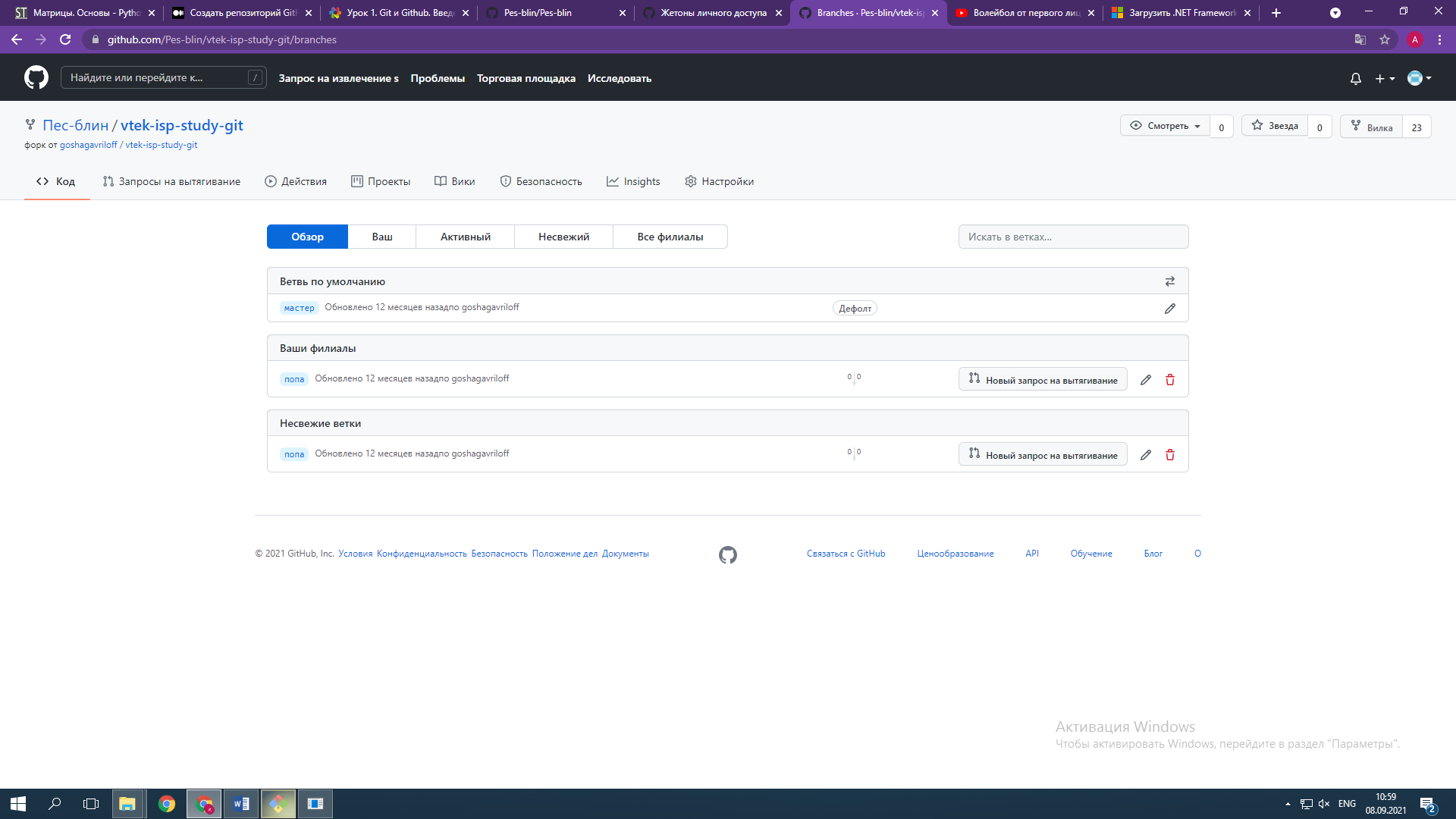
1. Зафиксируйте изменения (commit).

1. Создать feature branch (ветку) в локальном репозитории и пушить ее на сервер.



1. Убедитесь, что проект работает после ваших изменений.



1. Сделайте Pull Request.
2. Обсудите его с рецензентом в процессе Code Review. При необходимости, внесите изменения в свой Pull Request.
3. Перереключиться между feature branches.
4. Соединить ветки (бранч и основной truck репозитория) - операция merge.
5. Когда все довольны, Pull Request принимают — с этого момента ваши изменения попали в исходный репозиторий (upstream) и являются частью проекта.
6. Перереключиться между feature branches.
7. Соединить ветки (бранч и основной truck репозитория) - операция merge.
8. Получить историю ревизий (change log) по некоторому файлу.