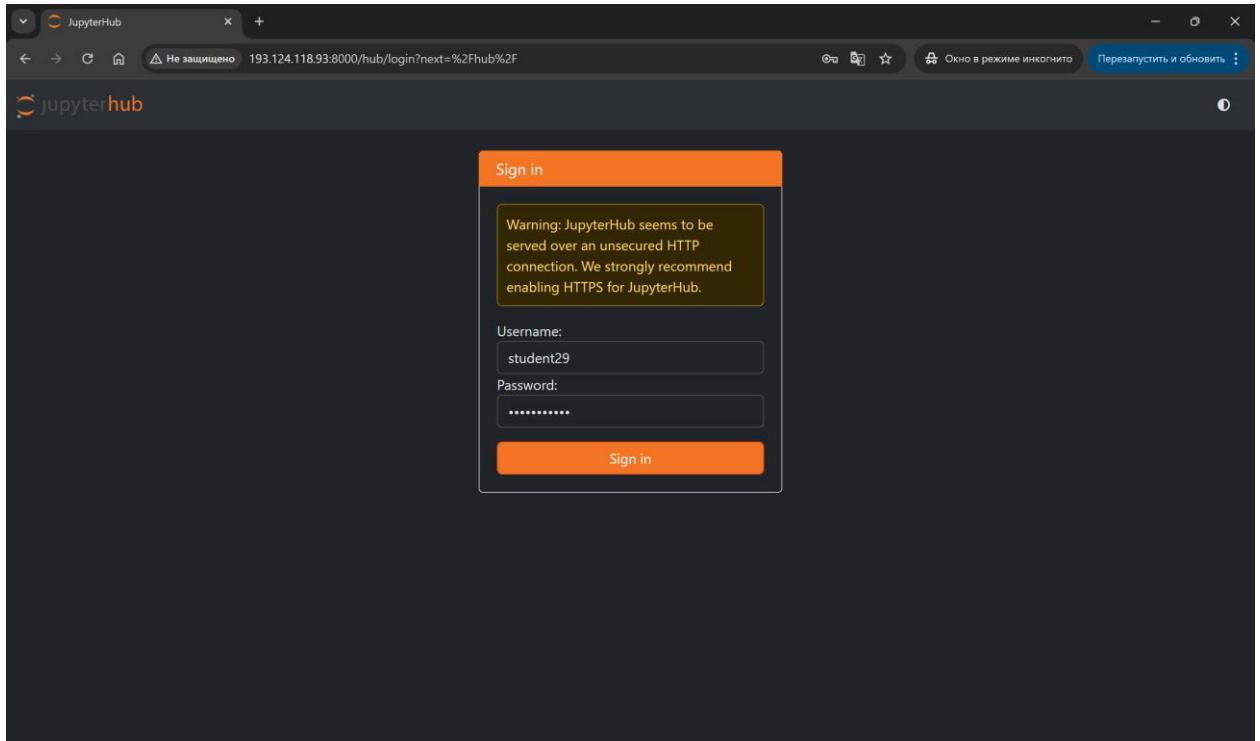
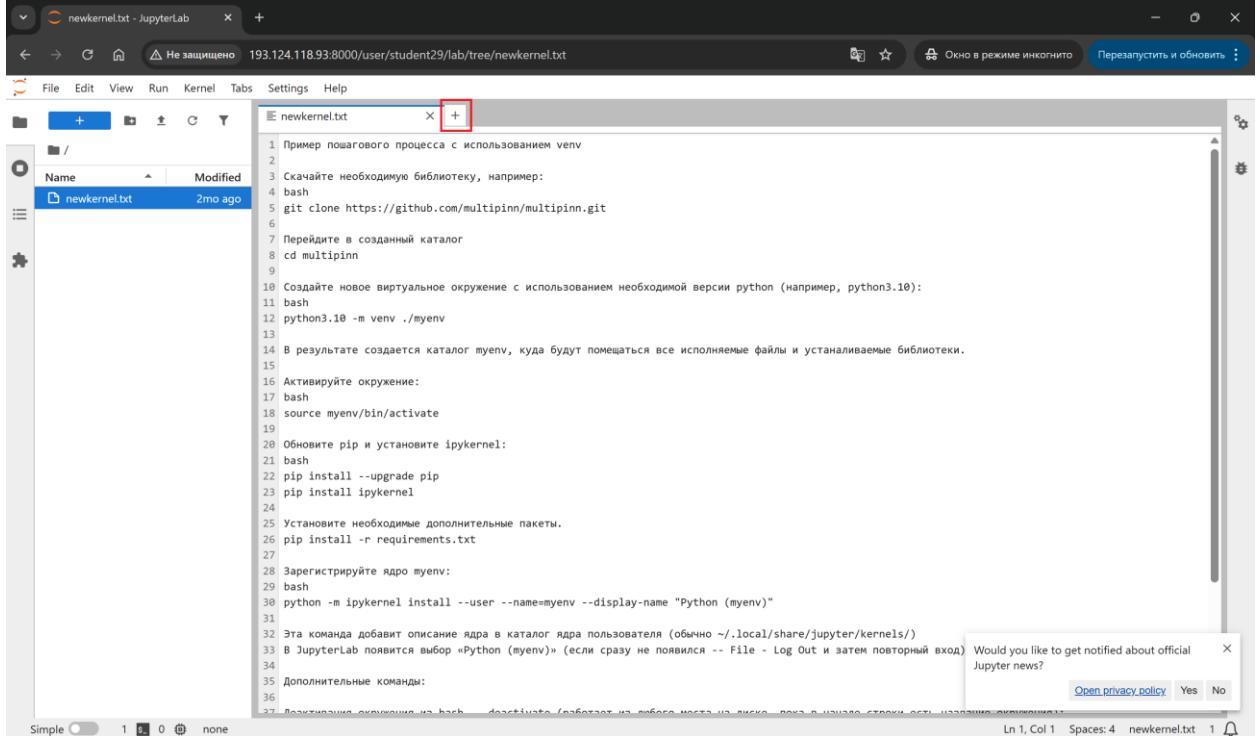


Настройка среды

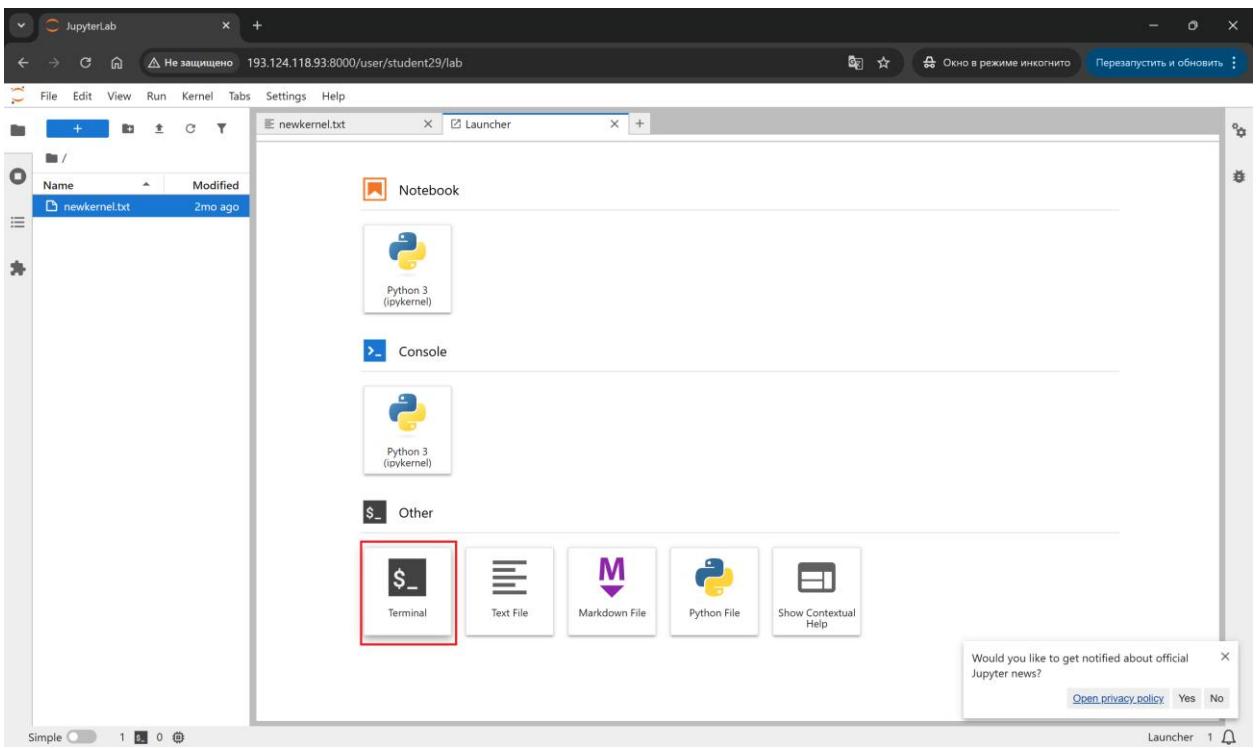
Авторизоваться на сервере [Jupyter-Hub](#) по адресу [Jupyter-Hub-ИИСТ-НПИ](#)



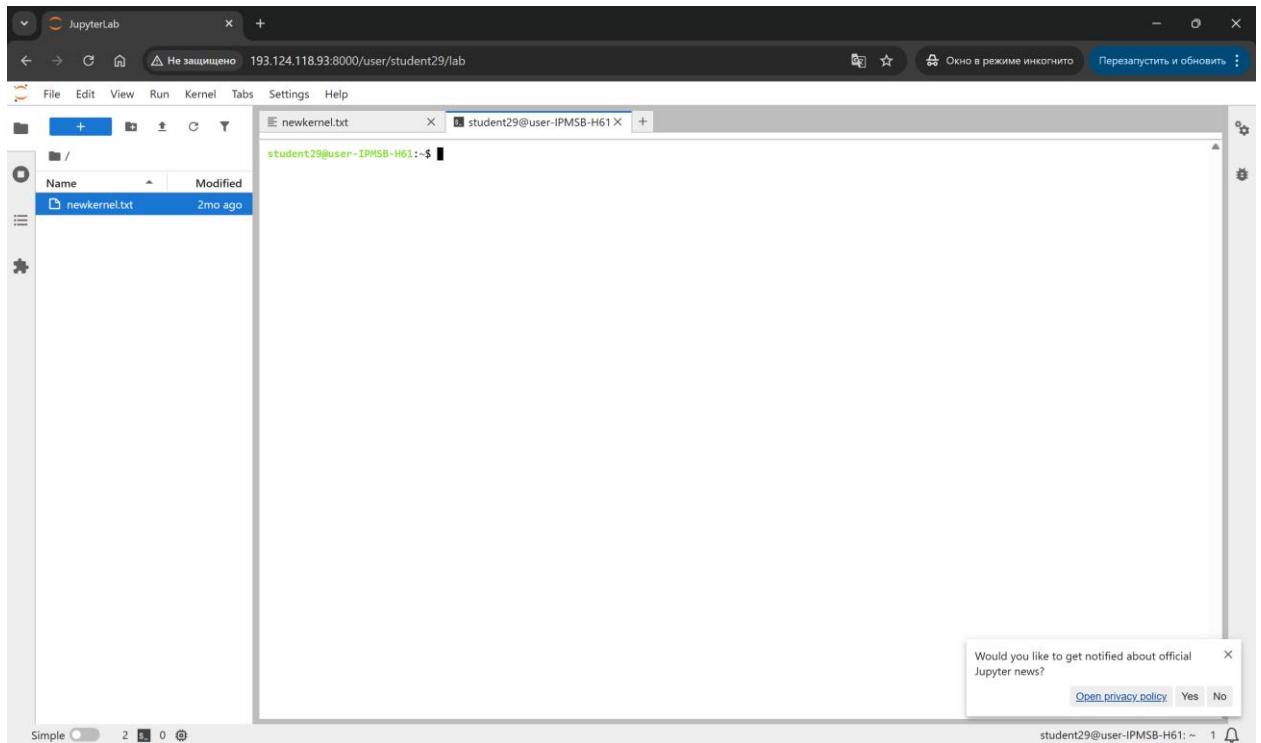
Создать новую вкладку символом +



Выбрать тип новой вкладки -- Terminal



Работать в новой вкладке вида



(При первом входе на сервер) Создать каталог с именем, соответствующим Вашим ФИО и году обучения, например:

```
mkdir ivanov_ii_2026
```

(При втором и последующих входах на сервер) Перейти каталог с именем, соответствующим Вашим ФИО и году обучения, например:

```
cd ivanov_ii_2026
```

Регистрация на Github

Зайдите на сайт GitHub, <https://github.com> - <https://github.com/join>

Определитесь с именем пользователя, введите адрес электронной почты, придумайте пароль, нажмите кнопку «Sign up for GitHub».

Далее необходимо пройти небольшой тест, чтобы доказать, что вы человек – нажмите «Verify» и решите предоставленную головоломку.

Затем нажмите «join a free plan», после чего вы попадете на экран, где сможете выбрать тип учетной записи — выберите бесплатную личную учетную запись (предлагается по умолчанию).

GitHub задает несколько вопросов о вашей работе, опыте программирования и интересах.

После этого вам необходимо ответить на полученное электронное письмо, подтверждающее, что вы связались с GitHub из учетной записи, к которой вы имеете доступ.

Далее вы можете создать репозиторий или организацию или пройти курс «Introduction to GitHub». Организация — это совокупность учетных записей пользователей, владеющих репозиториями. У организаций есть один или несколько владельцев, обладающих правами администратора для организации. Сердцем проекта на основе Git является репозиторий. Он содержит весь ваш код и связанные с ним артефакты, включая такие вещи, как:

- Файл README, описывающий цель проекта.
- Файл LICENSE, в которой необходимо указать способы использования вашего кода иными пользователями.

Вы можете сделать свой репозиторий частным (доступным только для людей с учетными записями, имеющими разрешение на его просмотр) или общедоступным (доступным для поиска и просмотра всеми).

Когда вы создадите свой репозиторий, вы заметите, что он имеет несколько вкладок и открывается на вкладке «Code». Code – место, где находятся все исходные файлы. Git изначально создавался как хранилище исходного кода, и теперь сюда попадают самые разные файлы.

Если вы выбрали автоматическое создание README и/или LICENSE, они также будут на данной вкладке.

Issues позволяет отслеживать открытые элементы в базе вашего проекта.

Pull Requests – это часть механизма взаимодействия с другими пользователями. Pull requests определяют изменения, которые зафиксированы и готовы к проверке перед объединением в основную ветку.

Projects – инструменты для управления, сортировки, планирования и т.д. ваших различных проектов.

Wiki, Security и Insights – эти инструменты часто оставляются для более опытных пользователей и обеспечивают основу для связи с внешним сообществом пользователей.

Settings – GitHub позволяет выполнять множество настроек, включая изменение имени вашего репозитория и контроль доступа.

Создание репозитория в GitHub

Нажмите «+», затем нажмите «New Repository».

Чтобы создать новый репозиторий, вам необходимо заполнить следующие данные:

- дать новому репозиторию имя;
- добавить описание репозитория;
- выбрать видимость репозитория – хотите ли вы, чтобы он был публичным или частным;
- выбрать опцию «Initialize this repository with a README».

Затем нажмите «Create Repository». Теперь вы будете перенаправлены в созданный вами репозиторий. Корневая папка вашего репозитория указана по умолчанию и содержит только один файл README.md.

Следующий шаг – отредактировать файл README. Это можно сделать в браузере: нажмите на карандаш, чтобы открыть онлайн-редактор, и вы сможете изменить текст файла README. Чтобы сохранить изменения в репозитории, вы должны зафиксировать их. После внесения изменений прокрутите вниз до раздела «Commit changes». Добавьте сообщение о фиксации и (при необходимости) описание, затем нажмите «Commit changes». «Commit changes» используется для сохранения изменений в репозитории. Вернитесь на главный экран, щелкнув ссылку на имя репозитория. Обратите внимание, что файл readme обновлен, и проверьте внесенные изменения.

Создание нового файла с помощью встроенного веб-редактора GitHub, который запускается в браузере. Нажмите «Add File», затем нажмите «Create New File», чтобы создать новый файл. Например, создаем файл Python с именем firstpython.py. Сначала укажите имя файла. Затем добавьте комментарий, описывающий ваш код, а затем добавьте код. После завершения зафиксируйте изменения в репозитории. Вы можете видеть, что ваш файл теперь добавлен в репозиторий, а в списке репозитория показано, когда файл был добавлен или изменен.

Если вам нужно изменить файл, необходимо выполнить следующее. Щелкните имя файла, а затем щелкните значок карандаша, внесите изменения и зафиксируйте их.

Вы также можете загрузить файл из вашей локальной системы в репозиторий. На главном экране репозитория нажмите «Add File» и выберите параметр «Upload files». Нажмите «Choose Your Files» и выберите файлы, которые вы хотите загрузить из своей локальной системы. Процесс загрузки файла может занять некоторое время, в зависимости от того, что вы загружаете. После завершения загрузки файлов нажмите «Commit Changes». В репозитории теперь отображаются загруженные файлы.

Создание branch

Чтобы добавить ветку в ваш репозиторий, выполните следующие шаги.

- Перейдите на главную страницу вашего репозитория. Обратите внимание, что когда вы создавали свой репозиторий, для вас была создана основная ветка.
- В верхней части списка файлов найдите раскрывающееся меню «branch». (По умолчанию в меню отображается «Branch: master».) Щелкните раскрывающееся меню, введите имя ветки, которую вы хотите создать, и нажмите Enter на клавиатуре.

В вашем репозитории теперь есть две ветки: `master` и `Child_Branch`. Вы можете щелкнуть раскрывающееся меню, чтобы увидеть свои ветки. Все файлы, которые находились в основной ветке, теперь скопированы в `Child_Branch`. Обратите внимание: когда вы добавляете или редактируете файл в `Child_Branch`, это изменение не будет автоматически внесено в основную ветку. Чтобы добавить файл в новую ветку, убедитесь, что `Child_Branch` (или любое другое имя, которое вы дали своей ветке) отображается в раскрывающемся меню «Branch», и выполните следующие шаги:

- Нажмите `Add file -> Create new file`, чтобы создать файл в репозитории.
- Введите имя и расширение файла — например, `testchild.py` — и добавьте какие-либо строки в тело нового файла.
- Прокрутите страницу вниз, добавьте описание файла, который вы собираетесь добавить (обратите внимание, что описание не является обязательным), и нажмите «Commit new file».

Файл, который вы добавили в дочернюю ветку, не добавляется автоматически в основную ветку. (Вы можете проверить это, используя раскрывающееся меню «Branch», чтобы перейти к основной ветке; обратите внимание, что в списке файлов нет файла `testchild.py`) Вы также можете сравнить две ветки и открыть запрос на включение, который позволит вам скопировать изменения, внесенные вами в дочернюю ветку (в данном случае добавление нового файла), в основную ветку.

- В `Child_Branch` нажмите кнопку «Compare & pull request».

- Прокрутите страницу вниз и обратите внимание, что в списке указан *1 changed file*.
- Прокрутите страницу вверх и обратите внимание, что GitHub сравнивает ветки *master* и *Child_Branch* и между ними нет конфликтов.

При желании вы можете добавить комментарий к запросу на включение. Нажмите *Create pull request*.

Операция «Merge» на базе «pull request»

Чтобы объединить ветки по запросу *pull request* в проекте, выполните следующие действия:

- Откройте вкладку «Pull requests». Отображается список ожидающих запросов на включение.
- Щелкните *pull request*, который вы хотите объединить с основным проектом. Если вас устраивают изменения, нажмите «Merge pull request», чтобы принять запрос на включение и объединить обновления (при желании вы можете добавить комментарий).
- При нажатии кнопки «Merge pull request» отображается кнопка «Confirm merge». Нажмите эту кнопку, чтобы завершить слияние.

pull request теперь успешно *merged*. Обратите внимание: вы можете удалить дочернюю ветку, поскольку ваши изменения были включены в *master* ветку.

Работа с локальным репозиторием посредством командной строки.

Возвращаемся в окно Terminal сервера, созданного на этапе [настройки](#)

Создаем новую директорию для локального репозитория:

- Создайте каталог *myrepo*, скопировав и вставив в терминал приведенную ниже команду *mkdir*:

```
mkdir myrepo
```

- Перейдите в каталог *myrepo*, скопировав и вставив команду *cd*:

```
cd myrepo
```

- В этом каталоге *myrepo* можно создать новый локальный репозиторий *git* с помощью команды *git init*. Скопируйте и вставьте в терминал команду:

```
git init
```

- Теперь создан новый локальный репозиторий, который вы можете проверить, выполнив список каталогов, вставив следующую команду в окно терминала:

```
ls -la .git
```

- Вывод показывает содержимое подкаталога `.git`, в котором находится локальный репозиторий.
- Теперь давайте создадим пустой файл, используя следующую команду: «».

```
touch newfile.txt
```

- Добавьте этот файл в репозиторий, используя следующую команду `git add`:

```
git add newfile.txt
```

- Прежде чем мы сможем зафиксировать наши изменения, нам нужно сообщить git, кто мы такие. Мы можем сделать это, используя следующие команды (вы можете скопировать эти команды как есть, без необходимости вводить фактическую информацию):

```
git config --global user.email "you@example.com"
```

```
git config --global user.name "Your Name"
```

- Как только в репозитории появится `newfile`, давайте зафиксируем наши изменения, используя команду `git commit`. Обратите внимание, что для фиксации требуется сообщение, которое мы включаем с помощью параметра `-m`:

```
git commit -m "added newfile.txt"
```

- Наш предыдущий коммит создал основную ветку по умолчанию под названием `master`. Чтобы внести последующие изменения в наш репозиторий, давайте создадим новую ветку в нашем локальном репозитории. Скопируйте и вставьте следующую команду `git branch` в терминал, чтобы создать ветку с именем `my1stbranch`:

```
git branch my1stbranch
```

- Давайте проверим, какие ветки содержит наше репо, вставив в терминал следующую команду `git branch`:

```
git branch
```

Обратите внимание, что в выводе перечислены две ветки – основная ветка по умолчанию со звездочкой * рядом с ней, указывающая, что это активная в данный момент ветка, и вновь созданная ветка `my1stbranch`.

- Поскольку теперь мы хотим работать в новой ветке, введите команду `git checkout`, чтобы сделать ее активной веткой для внесения изменений:

```
git checkout my1stbranch
```

- Давайте проверим, что новая ветка теперь является активной, выполнив следующую команду `git branch`:

```
git branch
```

Проверьте, что звездочка * теперь находится рядом с *my1stbranch*, указывая, что она теперь активна.

ПРИМЕЧАНИЕ. В качестве команды для создания новой ветки и перехода к ней с помощью git, вы можете также использовать следующую команду с опцией -b:

```
git checkout -b my1stbranch
```

- Давайте внесем некоторые изменения в вашу новую ветку под названием *my1stbranch*. Начните с добавления текста в *newfile.txt*, вставив в терминал следующую команду, которая добавит строку «Вот текст в моем новом файле» в файл:

```
echo 'Here is some text in my newfile.' >> newfile.txt
```

- Убедитесь, что текст добавлен, вставив следующую команду *cat*:

```
cat newfile
```

- Теперь давайте создадим еще один файл с именем *readme.md*, используя следующую команду:

```
touch readme.md
```

- Добавьте его в репозиторий с помощью следующей команды *git add*:

```
git add readme.md
```

На данный момент в нашей новой ветке мы отредактировали новый файл и добавили файл с именем *readme.md*. Мы можем легко проверить изменения в нашей текущей ветке с помощью команды *git status*.

```
git status
```

Вывод команды *git status* показывает, что файлы *readme.md* были добавлены в ветку и готовы к фиксации, поскольку мы добавили их в ветку с помощью *git add*. Однако, несмотря на то, что мы изменили файл с именем *newfile.txt*, мы не добавили его явно с помощью *git add*, и, следовательно, он не готов к фиксации. Команда для добавления всех изменений и дополнений – *git add* со звездочкой *; данный шаг также добавит измененный файл *newfile.txt* в ветку и подготовит его к фиксации:

```
git add *
```

- Давайте еще раз проверим статус:

```
git status
```

Вывод теперь показывает, что оба файла теперь могут быть зафиксированы.

- Теперь, когда наши изменения готовы, мы можем сохранить их в ветку, используя следующую команду фиксации с сообщением, указывающим изменения:

```
git commit -m "added readme.md modified newfile.txt"
```

- Мы можем выполнить следующую команду `git log`, чтобы получить историю последних коммитов.

```
git log
```

В журнале показаны два недавних коммита — последний коммит в *my1stbranch*, а также предыдущий коммит в *master*. Чтобы выйти из журнала `git`, введите `q`.

- Иногда вы не можете полностью протестировать свои изменения перед их фиксацией, и это может иметь нежелательные последствия. Вы можете отменить свои изменения, используя команду `git revert`, как показано ниже. Вы можете либо указать идентификатор вашего коммита, который вы можете увидеть в предыдущем выводе журнала, либо использовать ярлык `HEAD` для отката последнего коммита:

```
git revert HEAD --no-edit
```

- Если вы не укажете флаг `--no-edit`, вам может быть представлен экран редактора, показывающий сообщение с изменениями, которые необходимо отменить. В этом случае нажмите клавишу `Control` (или `Ctrl`) одновременно с `X`. Вывод показывает, что самый последний коммит с указанным идентификатором был отменен/
- Давайте внесем еще одно изменение в ваш текущий активный *my1stbranch*, используя следующие команды:

```
touch goodfile
```

```
git add goodfile
```

```
git commit -m "added goodfile"
```

```
git log
```

Вывод журнала показывает, что недавно добавленный *goodfile* был зафиксирован в *my1stbranch*.

- Теперь давайте объединим содержимое *my1stbranch* с основной веткой. Сначала нам нужно сделать активную ветку *master* с помощью следующей команды `git checkout`:

```
git checkout master
```

- Теперь давайте объединим изменения из *my1stbranch* в *master*:

```
git merge my1stbranch
```

```
git log
```

Вывод и журнал должны показать успешное слияние ветки.

- Теперь, когда изменения были объединены в главную ветку, *my1stbranch* можно удалить с помощью следующей команды `git branch` с опцией `-d`:

```
git branch -d my1stbranch
```

📌 Задание №1

- Создайте новую ветку под названием *newbranch*.
- Сделать новую ветку активной.
- Создайте пустой файл с именем *newbranchfile*.
- Добавьте вновь созданный файл в свою ветку.
- Зафиксируйте изменения в новой ветке.
- Отмените последние зафиксированные изменения.
- Создайте новый файл с именем *newgoodfile*.
- Добавьте последний файл в новую ветку.
- Зафиксируйте изменения.
- Объедините изменения в новой ветке с основной.

📌 Задание №2

Разбившись на подгруппы подвое выполнить:

- *fork* проекта второго студента;
- *clone* проекта в локальный репозиторий;
- добавление и фиксирование файла;
- синхронизацию с *fork*-репозиторием;
- формирование pull request к origin проекту на прием данного изменения. *ПРИМЕЧАНИЕ* для синхронизации потребуются команды `git push --set-upstream origin + имя новой ветки`, если таковая создавалась -> *merge* этих двух веток; если изменения выполнялись сразу в *master branch*, то используется простая команда `git push origin master`; для `git push` понадобится имя пользователя и не простой пароль пользователя, а *token (classic)*, полученный в соответствии с [инструкцией](#) (вводится вместо пароля в ходе выполнения `git push`).

📌 Подготовить отчет о выполненных Заданиях