|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение высшего образования* ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных технологий (МОСИТ)

|  |
| --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 1** |
| **по дисциплине** |
| **«Технологии разработки программных приложений»**  **Тема: «Системы контроля версий»** |

|  |  |
| --- | --- |
| Выполнил студент группы ИКБО-08-21 | Юнаков И.П. |
| Принял | Туманов М.Б. |

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2023

Оглавление

[Часть 1 3](#_Toc128402739)

[1. Установка и настройка клиента git 3](#_Toc128402740)

[2. Создание локального репозитория и добавление файлов 3](#_Toc128402741)

[3. Внесение изменений в файл 4](#_Toc128402742)

[4. Индексирование изменений и проверка состояния 4](#_Toc128402743)

[5. Создание коммита 5](#_Toc128402744)

[7. Просмотри истории коммитов 6](#_Toc128402745)

[8. Возврат к предыдущему состоянию 7](#_Toc128402746)

[9. Создание тегов 8](#_Toc128402747)

[10. Отмена изменений в рабочем каталоге 8](#_Toc128402748)

[11. Отмена коммитов в локальном репозитории 9](#_Toc128402749)

[Часть 2 11](#_Toc128402750)

[Часть 3 13](#_Toc128402751)

[Ответы на контрольные вопросы: 18](#_Toc128402752)

[1. Какие существуют типы систем контроля версий? Приведите примеры к каждому типу. 18](#_Toc128402753)

[2. К какому типу систем контроля версий относится Git? 18](#_Toc128402754)

[3. Что такое репозиторий Git? 18](#_Toc128402755)

[4. Что такое GitHub? 18](#_Toc128402756)

[5. Что такое коммит? 18](#_Toc128402757)

[6. Что такое ветка в репозитории Git? 19](#_Toc128402758)

[7. Что такое тег в репозитории Git? 19](#_Toc128402759)

[8. Что такое слияние двух веток? 19](#_Toc128402760)

# Часть 1

## **Установка и настройка клиента git**

* 1. Установить Git на свою рабочую станцию можно с помощью команды: sudo apt-get install git (для Ubuntu)
  2. Настроить имя пользователя можно с помощью команды: git config --global user.name " ilia yunakov "
  3. Настроить email можно с помощью команды: git config --global user.email "not\_really@mail.notrealy"
  4. Настроить игнорирование символов перевода строки можно с помощью команды: git config --global core.autocrlf input
  5. Включаем проверку преобразования, значение warn (печать только предупреждение, но принимает необратимый переход), команда**:** git config --global core.safecrlf warn

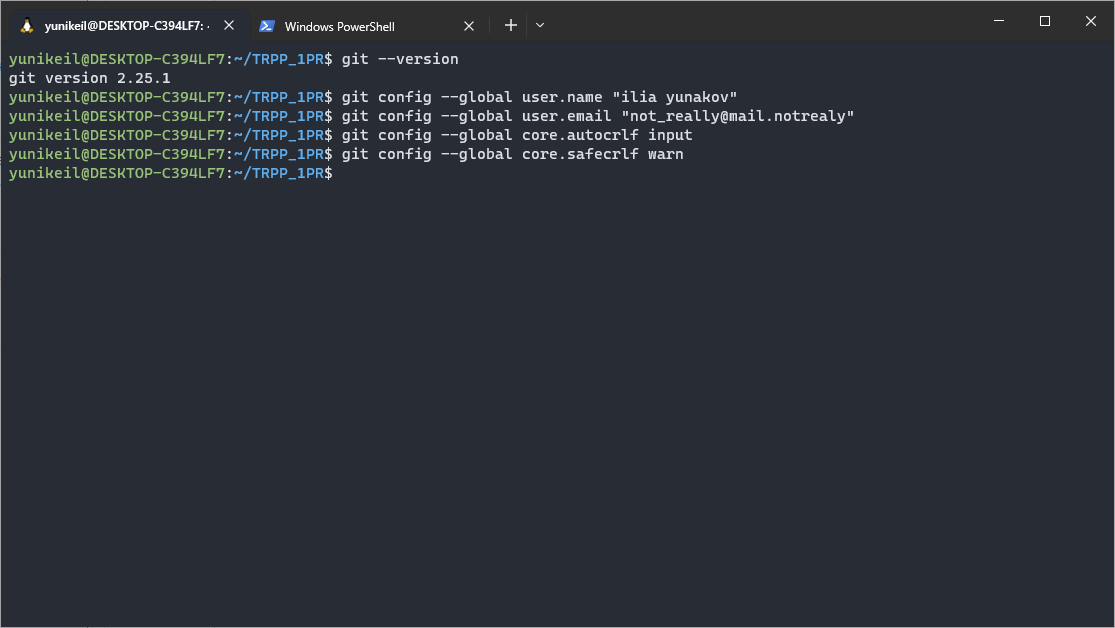


Рисунок 1 – Первый пункт первой части.

## **Создание локального репозитория и добавление файлов**

* 1. Создадим новую папку с помощью команды: mkdir myproject
  2. Перейдём в созданную папку с помощью команды: cd myproject
  3. Инициализируем локальный репозиторий с помощью команды: git init
  4. Создадим новый файл с помощью команды: touch myfile.txt
  5. Добавим созданный файл в индекс: git add myfile.txt

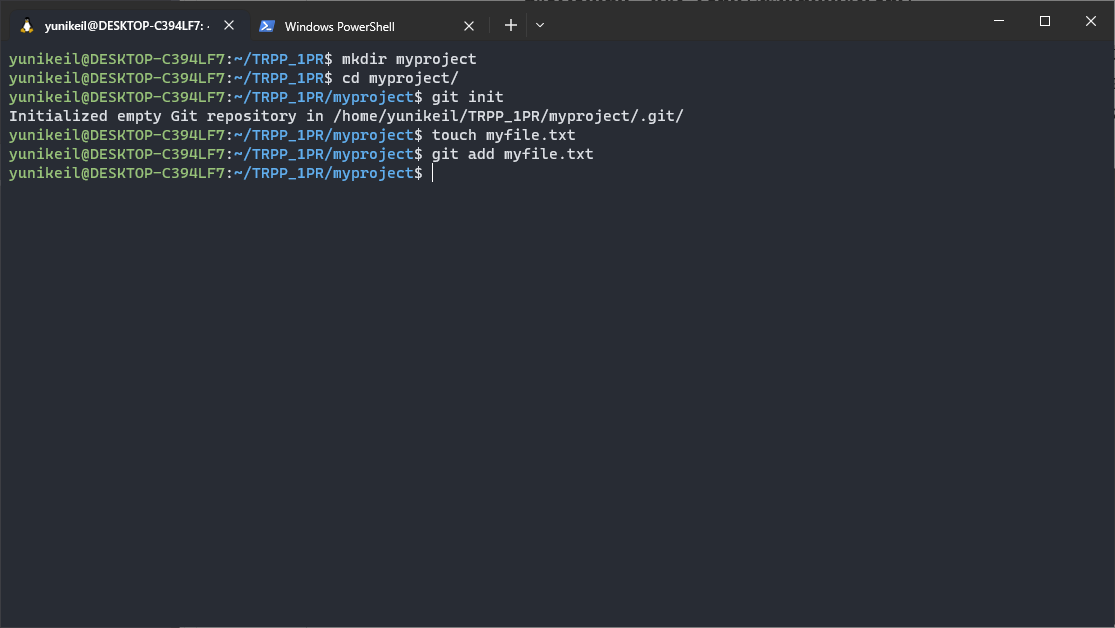


Рисунок 2 – Второй пункт первой части.

## **Внесение изменений в файл**

* 1. Откроем файл для редактирования с помощью команды: nano myfile.txt
  2. Внесём некоторые изменения в файл и сохраним его.

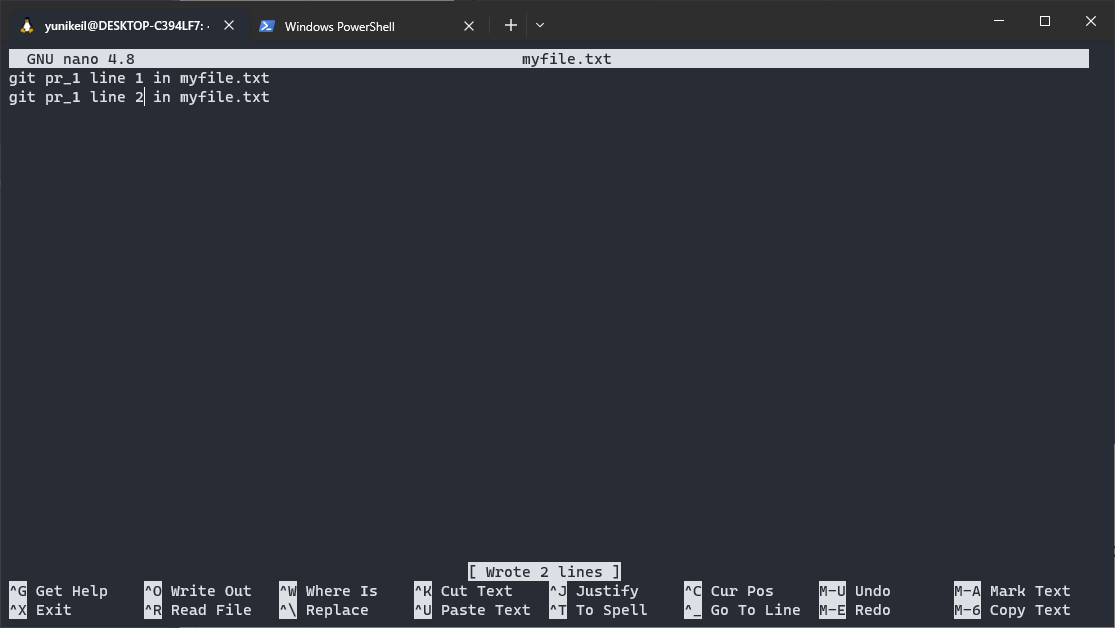


Рисунок 3 – Третий пункт первой части, Unix редактор nano.

1. Индексирование изменений и проверка состояния
   1. Для просмотра состояния файлов из рабочей директории можно использовать данную команду: git status
   2. Для того, чтобы проиндексировать изменения следует использовать данную команду: git add myfile.txt

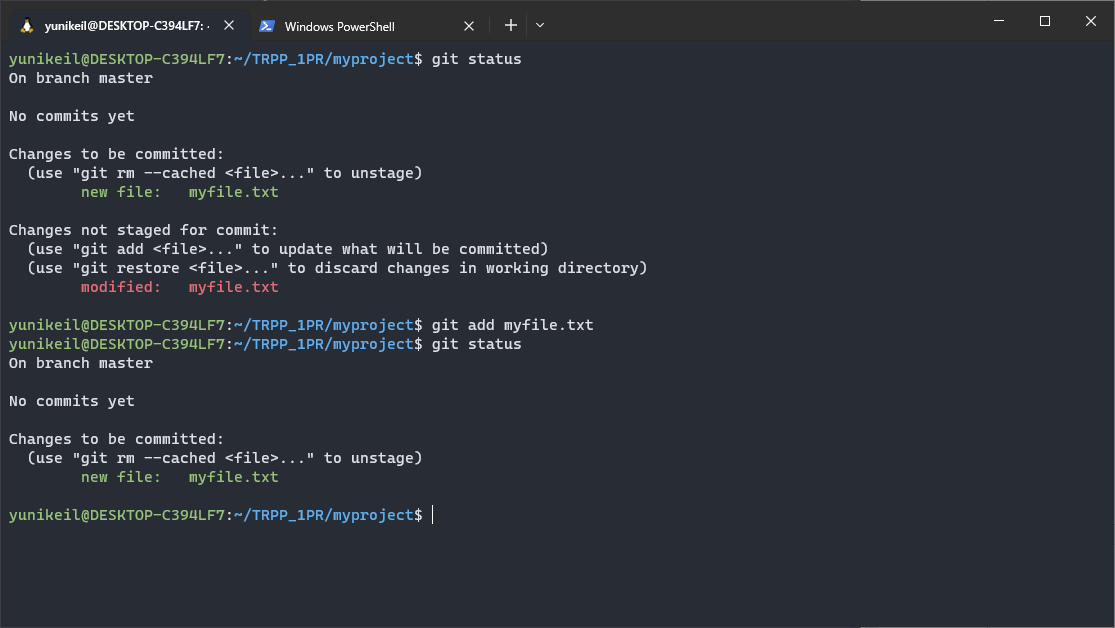


Рисунок 4 – Четвёртый пункт первой части.

1. Создание коммита
   1. Создание коммита происходит с помощью данной команды: git commit -m "Added a new lines to myfile.txt"



Рисунок 5 – Пятый пункт первой части.

1. Добавление ещё одного изменения в индекс, проверка состояния и создание второго коммита
   1. Для изменения файла нужно использовать команду: nano myfile.txt
   2. Для добавления изменений в индекс нужно использовать команду: git add myfile.txt
   3. Для изменения файла нужно использовать команду: nano myfile.txt
   4. Для проверки состояние нужно использовать команду: git status
   5. Для создания второго коммита следует использовать данную команду: git commit -m "Added a new line (3, 4) to myfile.txt"

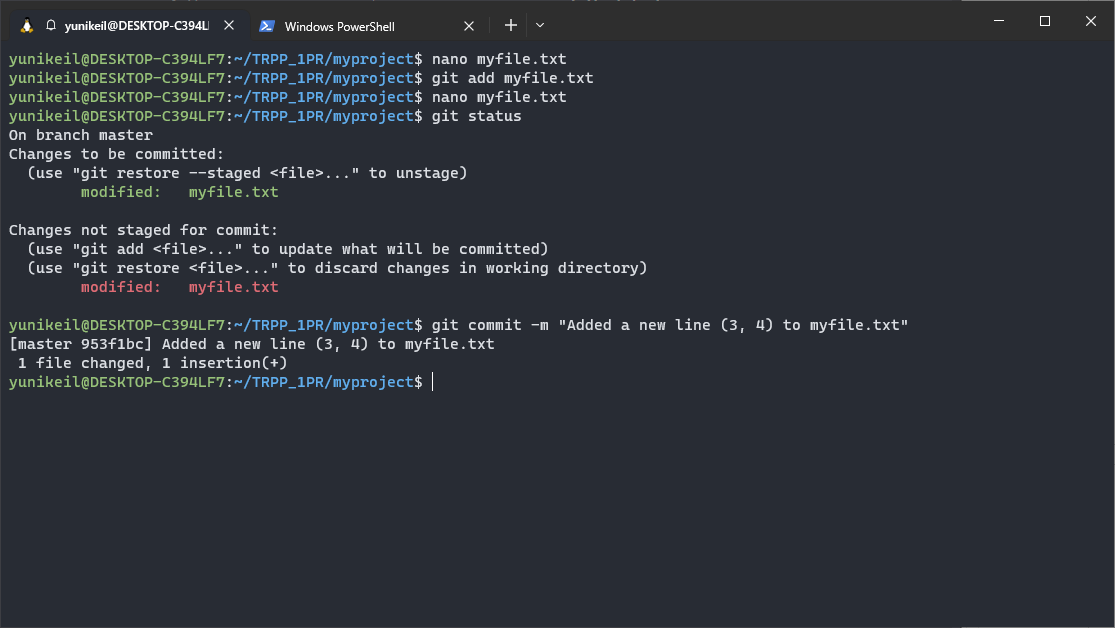


Рисунок 6 – Шестой пункт первой части.

1. Просмотри истории коммитов
   1. Для просмотри истории коммитов следует использовать команду: git log
   2. К команде из пункта 7.1. можно добавить аргументы, вот некоторые из них: --onelineили --graph

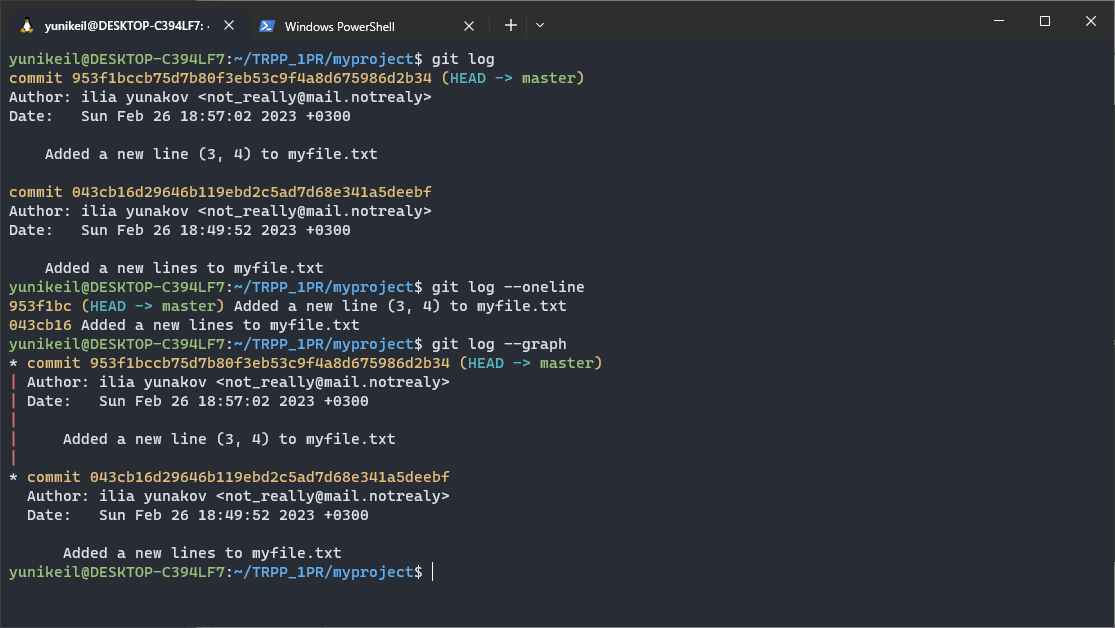


Рисунок 7 – Седьмой пункт первой части.

1. Возврат к предыдущему состоянию
   1. Для начала нужно просмотреть историю коммитов командой git log
   2. Далее нужно скопировать тег и откатить изменения к указанному коммиту с помощью команды git checkout <commit\_hash>

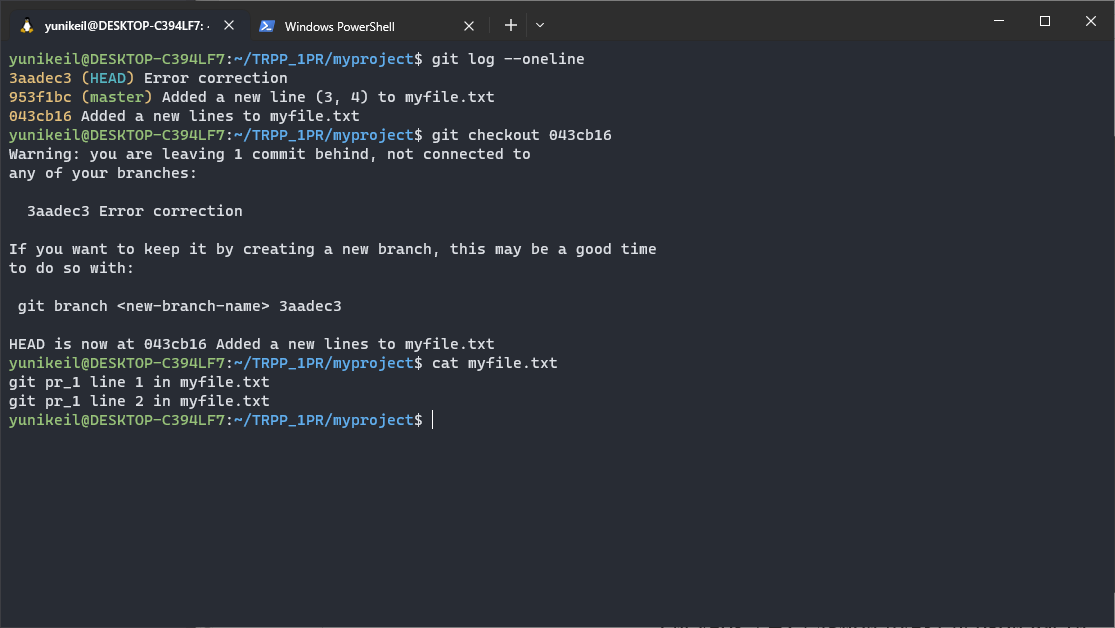


Рисунок 8 – Восьмой пункт первой части.

1. Создание тегов
   1. Чтобы создать тег для текущего коммита используется данная команда: git tag <tag\_name>
   2. Для просмотра списка тегов используется данная команда: git tag

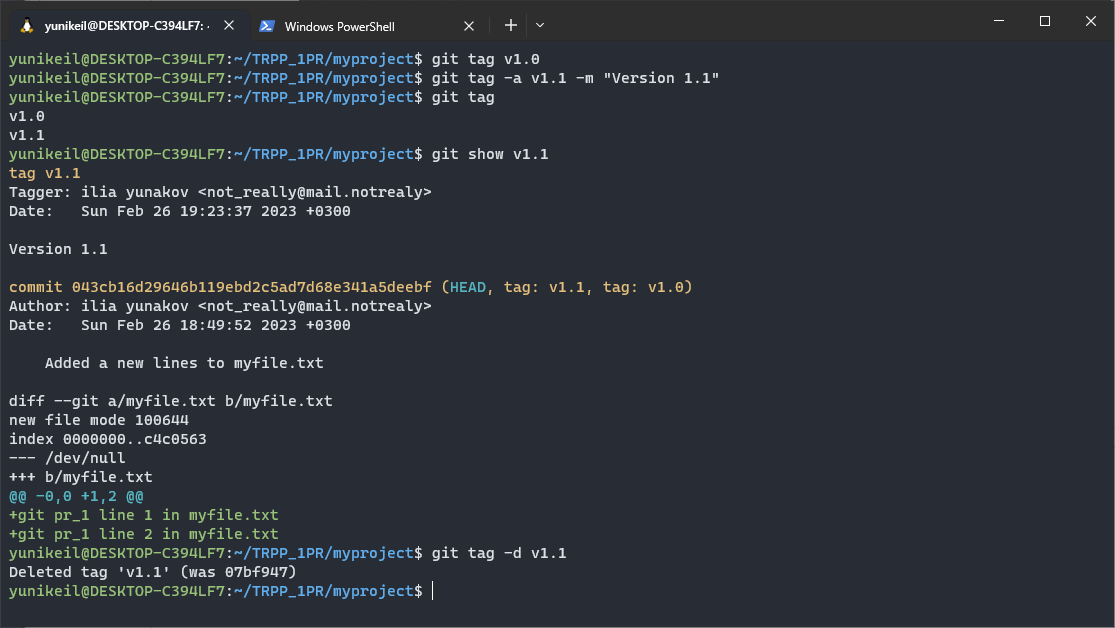


Рисунок 9 – Девятый пункт первой части.

1. Отмена изменений в рабочем каталоге
   1. Для отмены изменений, который не были проиндексированы стоит использовать команду git checkout -- myfile.txt
   2. Отменить изменения, которые были проиндексированы можно с помощью данной команды: git reset HEAD myfile.txt

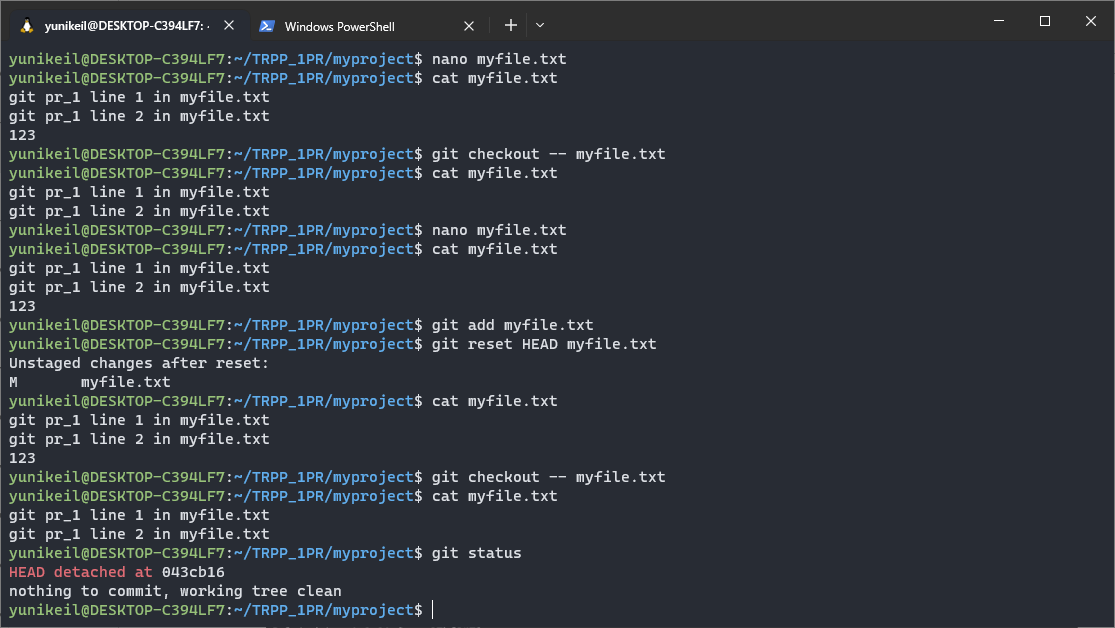


Рисунок 10 – Десятый пункт первой части.

1. Отмена коммитов в локальном репозитории
   1. Используем команду git log для нахождения хэша коммита, который хотим отменить
   2. Для создания нового коммита, который отменит изменения, внесённый в выбранный коммит стоит использовать команду git revert <commit-hash>. При этом исходный коммит сохраняется в истории.
   3. Для полного удаления коммита из истории можно использовать команду git reset. Однако использовать эту команду не рекомендуется, т.к. это может привести к проблемам синхронизации с другими разработчиками. Также можно добавить параметр --hard, данный параметр указывает на то, как изменять рабочее дерево после выполнения сброса. Если вы указали его, то все изменения, которые были внесены в рабочее дерево или проиндексированы в коммит, буду удалены.

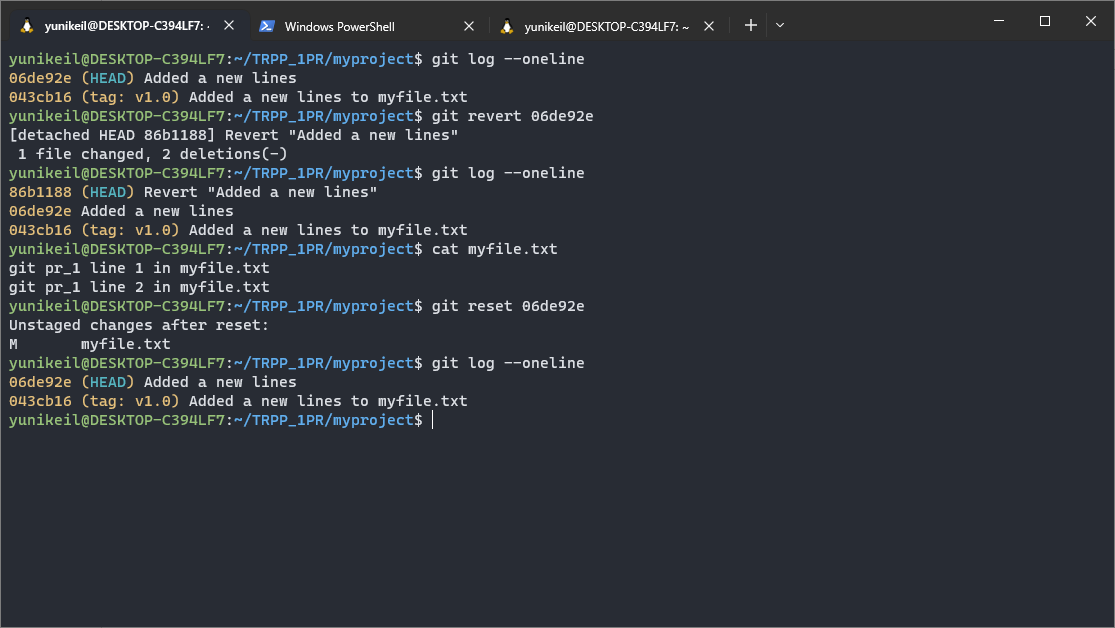


Рисунок 11 – Одиннадцатый пункт первой части.

# Часть 2

Курс на [сайте](https://learngitbranching.js.org/?locale=ru_RU) направлен на обучение работы с системой контроля версий Git и включает в себя следующие ключевые моменты:

1. Создание репозитория git init

mkdir my-project

cd my-project

git init

1. Использование команды git add для добавления изменений в индекс

git add file1.txt file2.txt

1. Использование команды git commit для фиксации изменений

git commit -m "Add file1.txt and file2.txt"

1. Использование команды git log для просмотра истории коммитов

git log

1. Создание новой ветки (branch) и переключение между ветками

git branch new-feature

git checkout new-feature

1. Использование команды git merge для объединения веток

git merge new-feature

1. Разрешение конфликтов при слиянии веток

# Внести необходимые изменения в конфликтующие файлы, затем добавить изменения в индекс и закоммитить

git add conflicted-file.txt

git commit -m "Resolve merge conflict"

1. Использование команды git rebase для изменения истории коммитов

# Переключиться на ветку, в которую необходимо включить изменения

git checkout main

# Изменить историю коммитов на ветке new-feature

git rebase new-feature

1. Использование команды git clone для клонирования репозитория

git clone https://github.com/username/my-project.git

1. Использование команды git push для отправки изменений на удаленный сервер

git push origin main

1. Использование команды git pull для получения изменений с удаленного сервера

git pull origin main

1. Работа с тегами (tags) для пометки важных версий репозитория

git tag v1.0.0

# Отправка тега на удаленный сервер

git push --tags

1. Работа с подмодулями (submodules) для включения других репозиториев в основной репозиторий

# Добавление подмодуля

git submodule add https://github.com/username/other-project.git

# Обновление подмодуля

git submodule update --remote

1. Работа с gitignore для игнорирования файлов при добавлении изменений в репозиторий.

# Создание файла .gitignore

touch .gitignore

# Добавление файлов, которые необходимо игнорировать

echo "node\_modules/" >> .gitignore

Эти ключевые моменты позволяют получить навыки работы с Git и осуществлять контроль версий своих проектов с помощью этой системы.

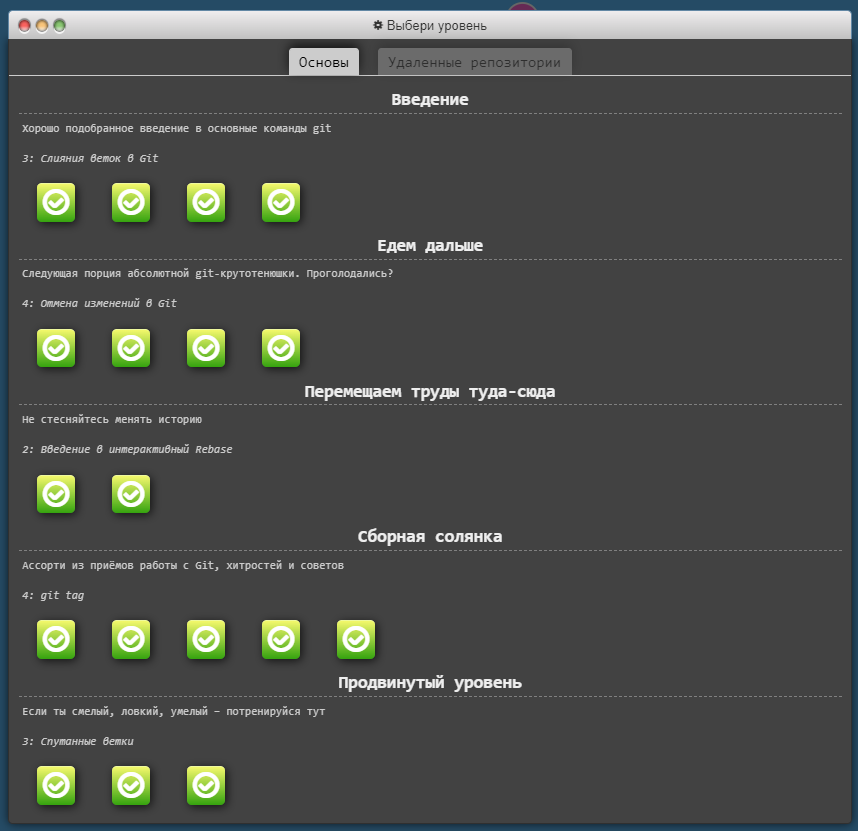


Рисунок 12 – Выполненные задания с сайта.

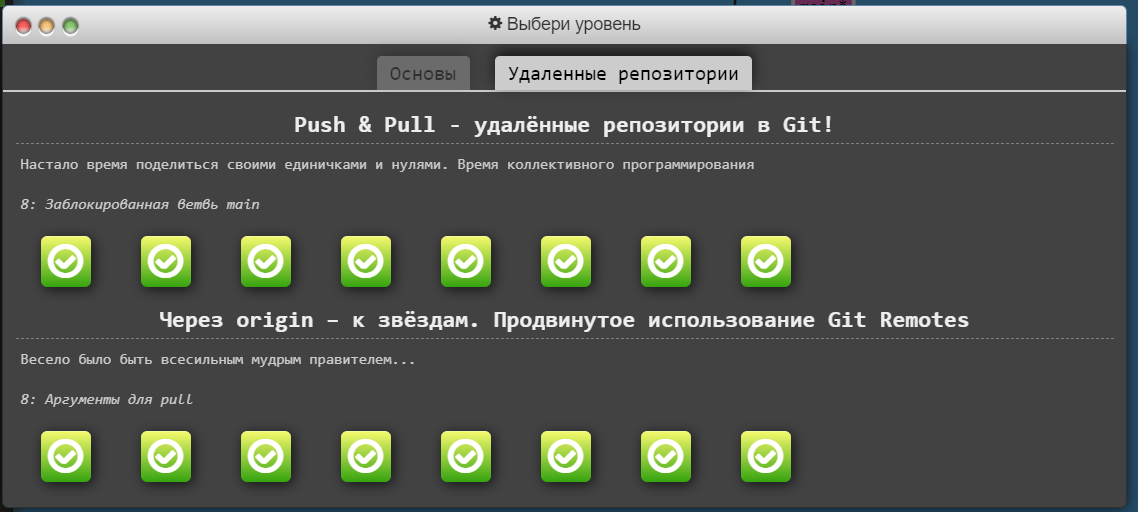


Рисунок 13 – Выполненные задания с сайта.

# Часть 3

1. Для создания аккаунта на GitHub нужно зайти на сайт, нажать “Sign up”, затем заполнить поля формы регистрации.
2. Для создания репозитория на GitHub нужно зайти в свой профиль на сайте, нажать на кнопку “New repository”, выбрать имя репозитория и задать настройки доступа. Далее нажать кнопку создать.
3. Для создания нового локального репозитория и его последующей загрузки на GitHub нужно выполнить следующие команды в терминале:
   1. Создаём папку проекта: mkdir my\_repo
   2. Перемещаемся в созданную папку: cd my\_repo
   3. Инициализируем локальный репозиторий: git init
   4. Создаём два пустых файла: touch file1.txt file2.txt
   5. Заполняем созданные файлы: echo "Hello, world!" > file1.txt
   6. Индексируем созданные файлы: git add .
   7. Коммитим изменения: git commit -m "Initial commit"
   8. Подробности настройки смотрите в п.4 данного списка. Настроенная команда для генерации ключа выглядит подобным образом: ssh-keygen -t rsa -b 4096 -C "notreal@mail.notreal”
   9. Создаём новое подключение на удалённом репозитории командой git remote add origin git@github.com:<username>/<repository>.git
   10. Выгрузим локальный репозиторий командой git push -u origin master

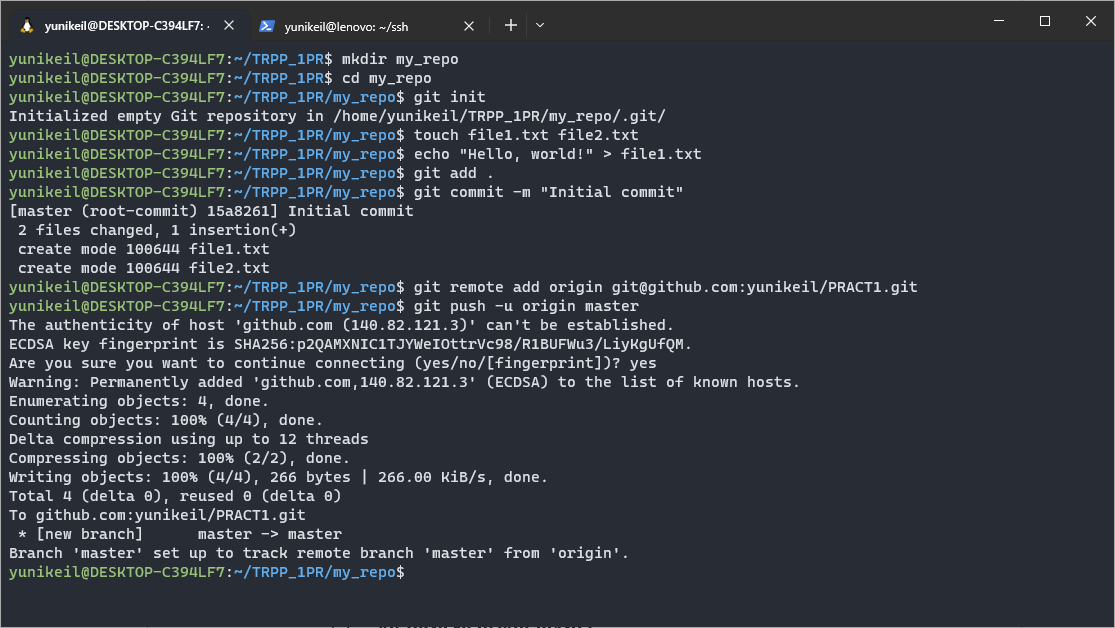
****

Рисунок 14 – Создание локального репозитория, его публикация на GitHub.

1. Для создания ключа требуется выполнить команду ssh-keygen, вытащить публичный ключ, добавить его в настройках аккаунта GitHub.
2. Для того, чтобы создать новую ветку в репозитории, с последующим слиянием в master нужно выполнить данные команды:
   1. git checkout -b new\_branch
   2. touch file3.txt
   3. echo "Hello again!" > file3.txt
   4. git add .
   5. git commit -m "Add file3.txt"
   6. git checkout master
   7. git merge new\_branch

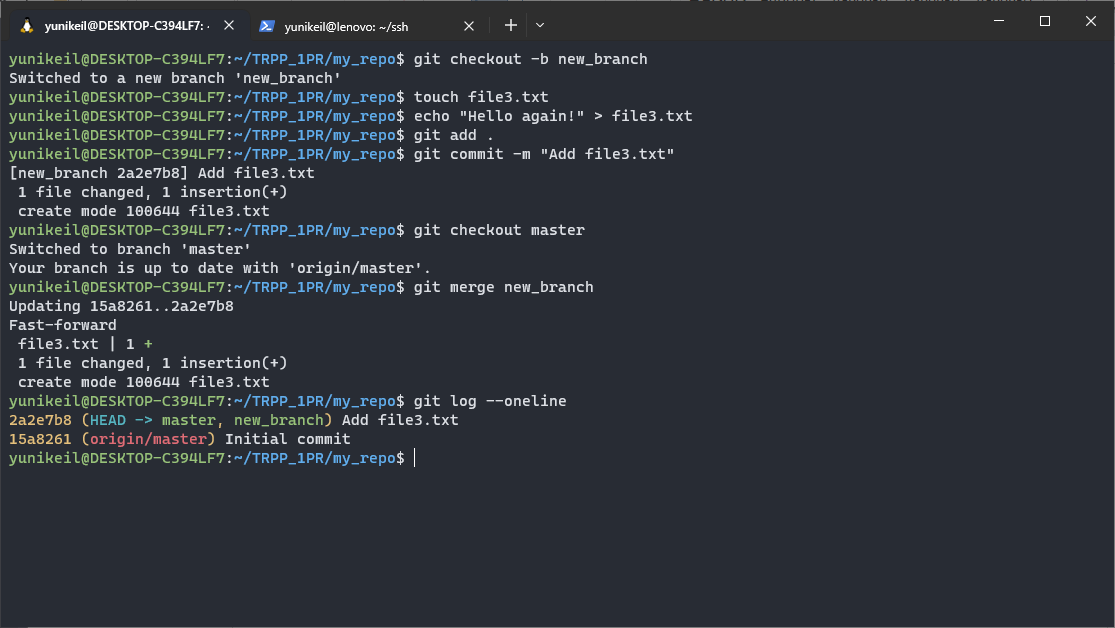
****

Рисунок 15 – Создание новой ветки, последующее слияние в master.

1. Вариант 13 << (30 - 1) % 15 - 1
   1. Удалим созданный ранее локальный репозиторий командой rm -r my\_repo, согласимся с удалением или добавим флаг f. Далее клонируем репозиторий командой git clone <url\_репозитория>. Я вместо url использовать ssh доступ по ключам, созданный в предыдущих шагах.
   2. Создадим новую ветку командой git checkout -b new\_branch, далее выведем список всех веток командой git branch -a
   3. Произведём 3 коммита в новой ветке с одним и тем же файлом используя команды echo, git add, git commit.
   4. Выгрузим изменения в удалённый репозиторий, используя команду: git push origin new\_branch
   5. Выведем информацию о том, в каком коммите была изменена каждая строка файла командой: git log -p file.txt
   6. Переключимся на ветку master, используя команду: git checkout master,обновим локальную ветку master, используя команду git pull origin master, для того чтобы убедиться, что она содержит последние изменения из удалённого репозитория. Далее используем merge в локальном репозитории, создаём pull request через менеджер проектов на сайте GitHub.

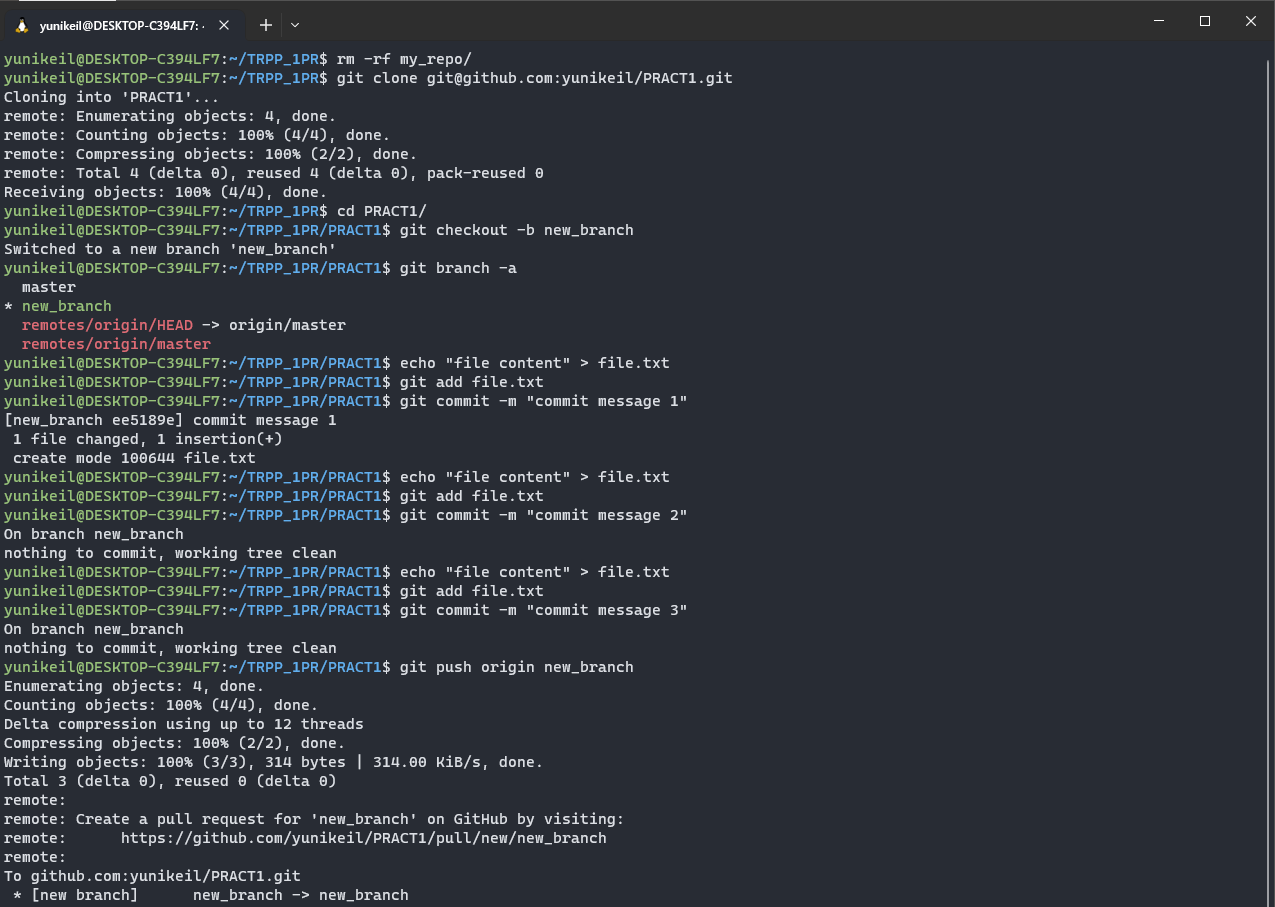


Рисунок 16 – Выполнение 6 задания.

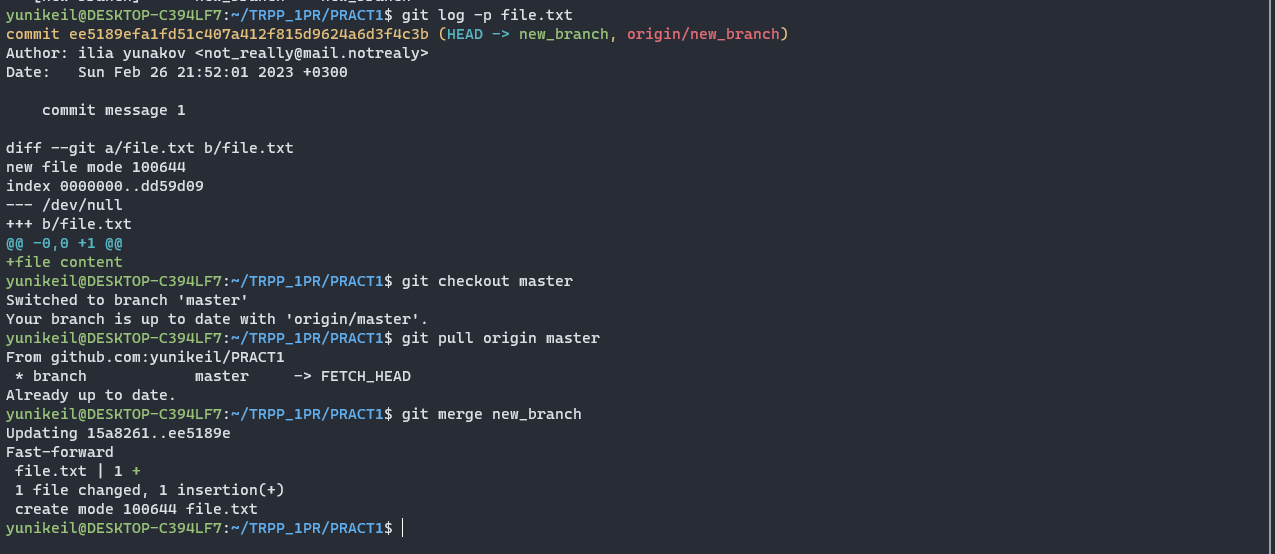


Рисунок 17 – Выполнения 6 задания.

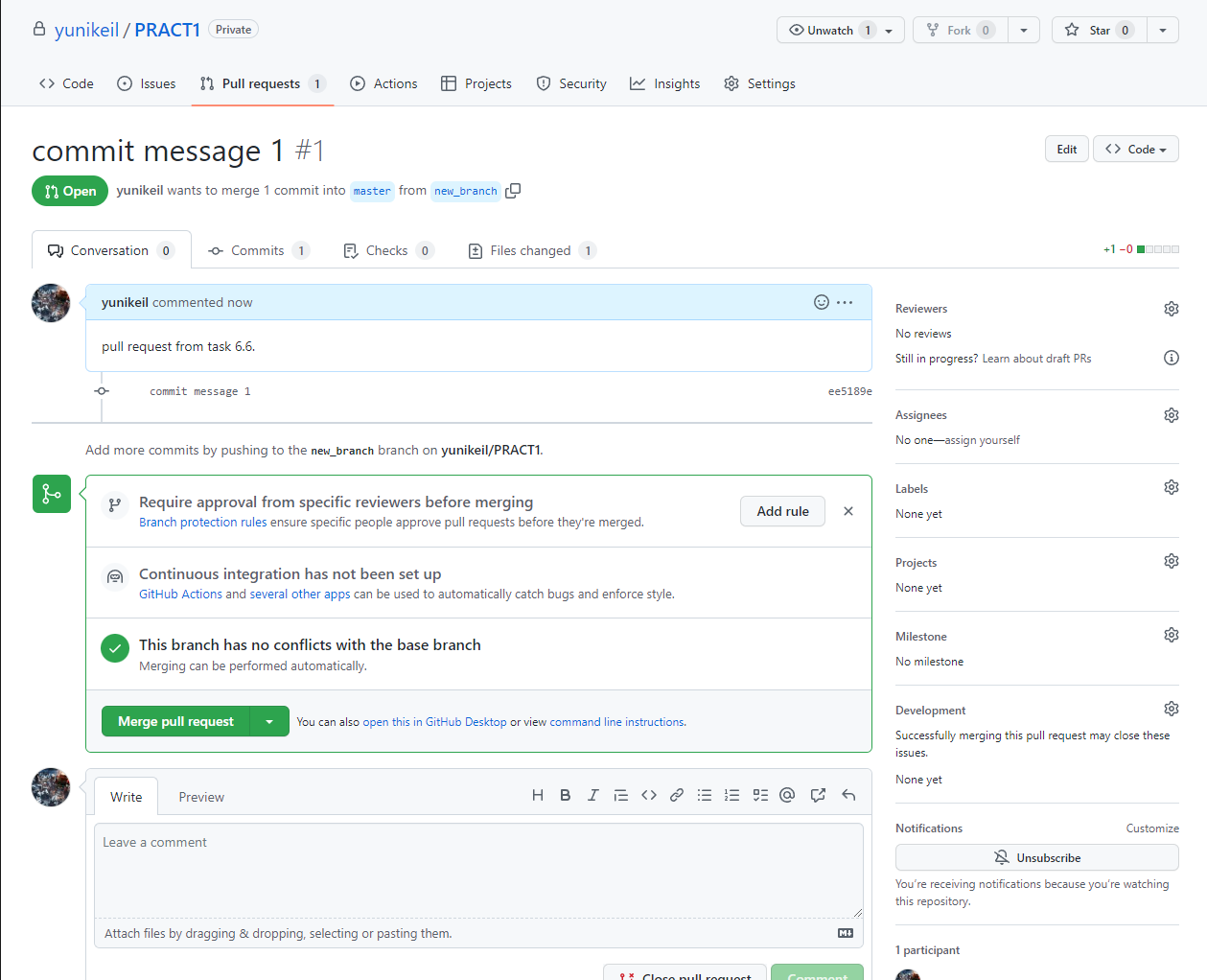


Рисунок 18 – Подтверждение запроса на вытягивание.

1. Вариант 8 << (30 - 1) % 10 - 1
   1. Создание форка происходит через меню на сайте GitHub.
   2. Клонирование репозитория по ssh: git clone git@github.com:yunikeil/domain-driven-hexagon.git
   3. Создаём две ветки brach1и brach2от последнего коммита в master’е.
   4. Проводим по 3 коммита в каждую из веток, которые меняют один и тот же кусочек файла.
   5. Выполняем слияние ветки branch1 в ветку branch2, разрешив конфликты при этом.
   6. Выгружаем все изменения во всех ветках в удалённый репозиторий.
   7. Проводим ещё 3 коммита в ветку branch1.
   8. Клонируем репозиторий ещё раз в другую директорию.
   9. В новом клоне репозитория создаём 3 коммита в ветку branch1.
   10. Выгружаем все изменения из нового репозитория в удалённый репозиторий.
   11. Возвращаемся в старый клон, выгружаем изменения командой pull --force
   12. Получаем все изменения в новом репозитории

# Ответы на контрольные вопросы:

1. Какие существуют типы систем контроля версий? Приведите примеры к каждому типу.

Существует несколько типов систем контроля версий:

Централизованные (Centralized) – все изменения хранятся на центральном сервере, а пользователи получают копии на свои компьютеры для работы с ними. Примеры: Subversion (SVN), Perforce.

Распределенные (Distributed) – каждый пользователь получает полную копию репозитория на свой компьютер, и изменения хранятся на каждом из них. Примеры: Git, Mercurial.

Локальные (Local) – система контроля версий, встроенная в IDE, работает только на одном компьютере. Примеры: Revision Control System (RCS), Source Code Control System (SCCS).

Пример создания репозитория Git:

$ git init myrepo

Initialized empty Git repository in /path/to/myrepo/.git/

1. К какому типу систем контроля версий относится Git?

Git относится к типу распределенных систем контроля версий.

1. Что такое репозиторий Git?

Репозиторий Git — это хранилище всех файлов и метаданных, необходимых для управления версиями проекта. Он содержит все версии всех файлов проекта и информацию о том, кто и когда их менял.

Пример клонирования удаленного репозитория в Git:

$ git clone https://github.com/username/repo.git

1. Что такое GitHub?

GitHub – это онлайн-платформа для хостинга Git-репозиториев, которая позволяет пользователям хранить, управлять и совместно работать над своими проектами.

1. Что такое коммит?

Коммит — это запись изменений в репозитории, которая содержит информацию о том, какие файлы были изменены, что было изменено.

Пример создания коммита в Git:

$ git commit -m "Added new feature"

1. Что такое ветка в репозитории Git?

Ветка в репозитории Git — это отдельная линия разработки, которая может отличаться от основной линии. Ветки позволяют разрабатывать разные функции или исправлять ошибки, не затрагивая основную линию разработки.

Пример создания новой ветки в Git:

$ git branch feature

1. Что такое тег в репозитории Git?

Тег в репозитории Git — это метка, которая указывает на определенный коммит. Теги обычно используются для указания релизных версий проекта.

Пример создания тега в Git:

$ git tag v1.0

1. Что такое слияние двух веток?

Слияние в Git – это процесс объединения изменений, внесенных в две различные ветки, в единую ветку. Обычно слияние используется для интеграции изменений из одной ветки в другую, когда работа в обеих ветках производится параллельно.

Процесс слияния может быть автоматическим или конфликтным. Автоматическое слияние возможно, когда Git может автоматически объединить изменения в двух ветках без конфликтов. Однако, если две ветки содержат изменения в одной и той же части кода, то возникает конфликт слияния, который должен быть решен вручную.

Пример кода для слияния двух веток в Git:

echo “#Переключаемся на ветку, в которую будем сливать изменения”

git checkout main

echo “выполняем команду для слияния ветки feature в текущую ветку”

git merge feature

Если Git не может выполнить автоматическое слияние изменений и возникает конфликт слияния, Git отмечает файлы, в которых есть конфликты, как "unmerged". Для решения конфликта необходимо внести изменения вручную в каждый из этих файлов, разрешив конфликты. После того, как конфликт будет разрешен, необходимо добавить изменения и зафиксировать коммит, чтобы завершить слияние.

Пример кода для разрешения конфликта слияния в Git:

echo "после выполнения команды merge возник конфликт в файле some\_file.py"

echo "файл помечен как unmerged"

git status

echo "открываем файл some\_file.py и разрешаем конфликты"

echo "после внесения изменений сохраняем файл"

echo "добавляем изменения в Git"

git add some\_file.py

echo "зафиксировываем коммит с разрешением конфликта"

git commit -m "Resolved merge conflict in some\_file.py"

Листинг 1 – Код для выполнения 7 задания.

1. #!/bin/bash
2. rm -rf domain-driven-hexagon domain-driven-hexagon2
3. clear
4. echo "Пункт 1. Склонировано с https://github.com/yunikeil/domain-driven-hexagon"
5. echo "Пункт 2. Клоинрование репозитория"
6. git clone git@github.com:yunikeil/domain-driven-hexagon.git
7. cd domain-driven-hexagon
8. echo "Пункты 3-4. Проведите по три коммита в двух ветках branch1, branch2"
9. # Создаем ветку branch1 и переключаемся на нее
10. git checkout master
11. git pull
12. git checkout -b branch1
13. # Цикл для создания трех коммитов в ветке branch1
14. for i in {1..3}
15. do
16. echo "Commit $i in branch1" >> README.md
17. git add README.md
18. git commit -m "Commit $i in branch1"
19. done
20. # Переключаемся на ветку branch2
21. git checkout -b branch2
22. # Цикл для создания трех коммитов в ветке branch2
23. for i in {4..6}
24. do
25. echo "Commit $i in branch2" >> README.md
26. git add README.md
27. git commit -m "Commit $i in branch2"
28. done
29. echo "Пункт 5. Выполните слияние ветки branch1 в ветку branch2, разрешив конфликты при этом"
30. git checkout branch2
31. git merge branch1
32. # Возможно, потребуется разрешить конфликты
33. echo "Пункт 6. Выгружаем все изменения во всех ветках в удалённый репозиторий"
34. git push origin --all
35. echo "Пункт 7. Проведём еще 3 коммита в ветку branch1"
36. git checkout branch1
37. # Цикл для создания трех коммитов в ветке branch1
38. for i in {7..9}
39. do
40. echo "Commit $i in branch1" >> README.md
41. git add README.md
42. git commit -m "Commit $i in branch1"
43. done
44. echo "Пункт 8. Склонировать репозиторий еще раз в другую директорию"
45. cd ..
46. git clone git@github.com:yunikeil/domain-driven-hexagon.git domain-driven-hexagon2
47. cd domain-driven-hexagon2
48. echo "Пункт 9. В новом клоне репозитории сделать 3 коммита в ветку branch1"
49. git checkout branch1
50. # Цикл для создания трех коммитов в ветке branch1
51. for i in {10..12}
52. do
53. echo "Commit $i in branch1" >> README.md
54. git add README.md
55. git commit -m "Commit $i in branch1"
56. done
57. echo "Пункт 10. Выгрузить все изменения из нового репозитория в удалённый рпеозиторий"
58. git push origin --all
59. echo "Пункт 11. Вернитесь в старый клон с репозиторием, выгрузите изменения командой pull с опцией --force"
60. cd ../domain-driven-hexagon
61. git pull --force origin
62. echo "Пункт 12. Получите все изменения в новом репозитории"
63. cd ../domain-driven-hexagon2
64. git pull