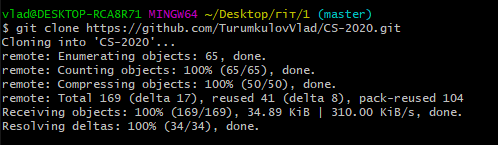
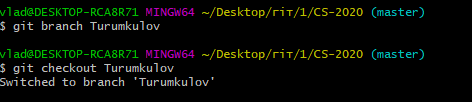
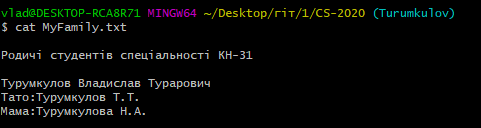
1. Клонуємо репозиторій CS-2020.



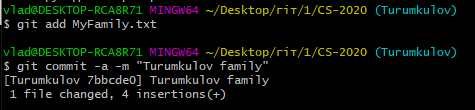
1. Створюємо гілку Turumkulov та переходимо в неї.



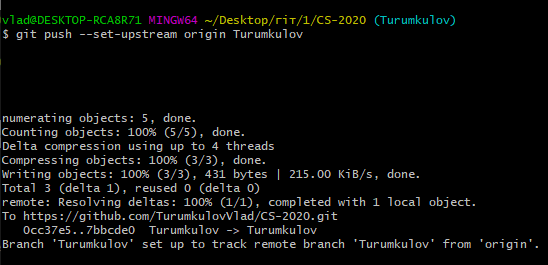
1. Вносимо зміни в файл MyFamily.txt та зберігаємо файл.



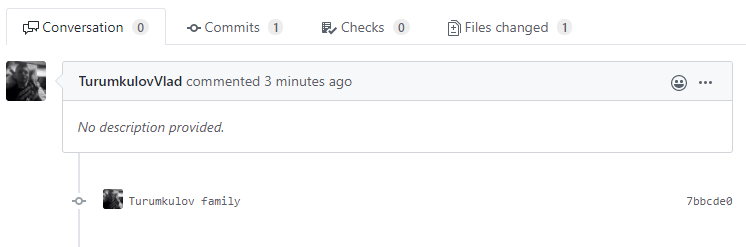
1. Робимо коміт змін.



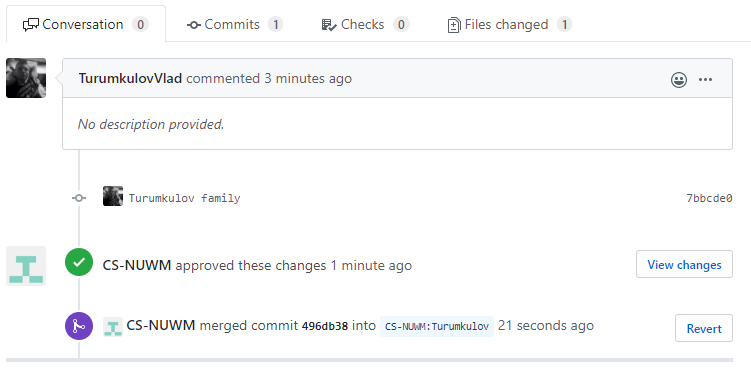
1. Зберігаємо внесені зміни до гілки Turumkulov глобального репозиторію.



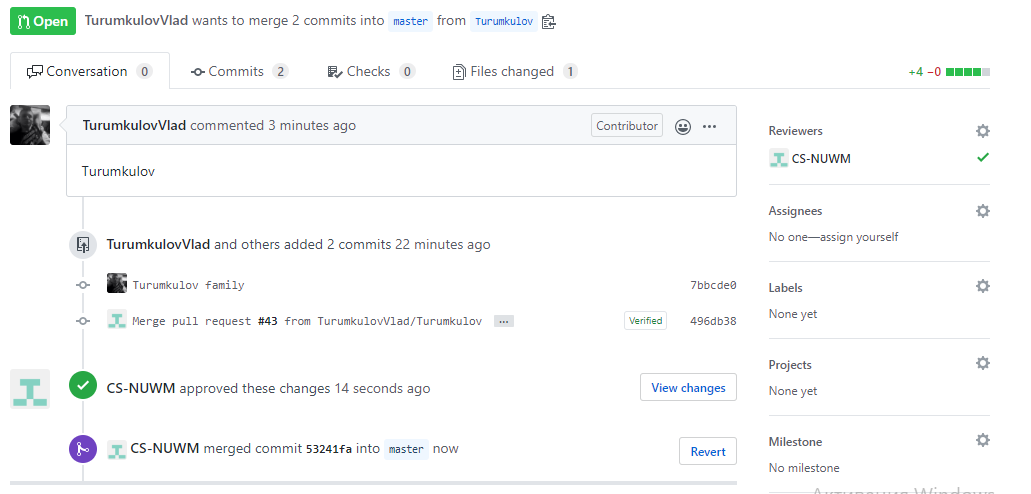
1. Зберігаємо зміни з TurumkulovVlad: Turumkulov до гілки CS-NUWM: Turumkulov



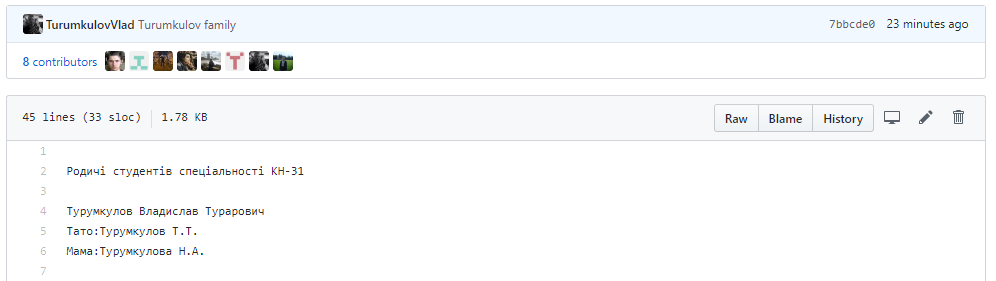
1. Створюємо запит на додавання файлу MyFamily.txt до гілки master



1. Злиття гілки Turumkulov та гілки master.



1. Результат злиття гілок.



**Контрольні питання**

1. **Основні гілки (master)**

Багато розробників підтримують такий процес, коли основна гілка містить стабільну версію коду — найімовірніше того, що був чи буде запроваджений у виробництво. Часто СКВ мають спеціальні назви для основної гілки. Наприклад, «master» в Git, або trunk в SVN.

Робочі гілки (develop)

Паралельно до основної гілки існує ще робоча гілка (англ. release branch). Вона використовуються для розробки нової версії програмного забезпечення та тестування його стабільності. Коли команда розробників вирішує, що нова версія програми готова до випуску, зміни з робочої гілки переносять у основну (master) і позначаються номером версії. Робочі гілки зазвичай зберігаються протягом тривалого часу.

**Додаткові гілки**

Поряд з основними гілками — master і develop, можливе існування також додаткових гілок, які забезпечують паралельну розробку проекту декількома членами команди, полегшують відслідковування помилок, а також сприяють швидкому усуненню поточних проблем, які виникатимуть у процесі розробки проекту. У процесі розробки складного проекту можуть використовуватися різні види додаткових гілок, зокрема:

**Тематичні гілки (feature)**

**Гілки версій (release)**

**Гілки помилок (hotfix)**

**Тематичні гілки (feature)**

Тематичні гілки є відгалуженням від робочої гілки (develop) і після завершення розробки визначеного функціоналу на них повинні бути злиті саме з робочою гілкою. Тематичні гілки використовуються для розробки нових, специфічних функцій майбутньої версії проекту. Основною особливістю даних гілок є те, що вони існують доки триває розробка певної функції, а потім після злиття з гілкою develop вони видаляються.

**Гілки версій (release)**

Гілки версій призначені для підготовки нової версії проекту. Вони дозволяють за мить до завершення роботи над поточною версією визначати місця підключення до головної гілки, а також надають можливість виконувати незначні правки й готувати дані про версію (номер, дата створення тощо). Виконання такого виду операцій на гілці версії дозволяє значно спростити розгалуження головної гілки develop. Відгалуження гілки версії від робочої гілки develop відбуватиметься за умови отримання бажаного стану нової версії проекту на робочій гілці.

**Гілки помилок (hotfix)**

Гілки помилок функціонально дуже схожі на гілки версій, оскільки також використовуються для підготовки нової версії проекту, але створюються вони спонтанно. Утворюються гілки помилок у разі потреби виконання миттєвих дій задля усунення небажаних станів робочої версії проекту. Тобто, якщо потрібно виправити критичну помилку у робочій версії проекту, то створюється відгалуження гілки помилок від головної гілки master, на тому її етапі, що відповідає робочій версії проекту в цей момент часу. Суттю використання таких гілок є те, що у разі виникнення непередбачуваних помилок робота членів команди на головній гілці develop не буде зупинена, а помилки будуть швидко знайдені і виправлені.

1. Для авторизації в обліковий запис GitHub, та можливості виконувати команду git push.
2. **Протоколи HTTP**

Git може спілкуватись через HTTP у двох режимах. До версії 1.6.6 у Git був тільки один метод, що був дуже простим та зазвичай тільки для читання. У версії 1.6.6 та новіших, був доданий кмітливіший протокол, що включає можливість Git проводити розумну передачу даних, схожу на те, як він працює через SSH. В останні декілька років, цей новий протокол HTTP став дуже розповсюдженим, адже він простіший для користувача та кмітливіший щодо методу передачі. Новішу версію часто називають **розумним** HTTP протоколом, а старішу — **тупим** HTTP Ми спочатку поговоримо про Розумний HTTP протокол.

**Розумний HTTP**

Розумний HTTP діє дуже схоже на те, як працює SSH чи Git протоколи, проте працює через звичайні HTTPS порти та може використовувати різноманітні механізми авторизації HTTP, отже часто цей метод простіший для користувачів, ніж SSH, адже можна використовувати логін/пароль авторизації замість налаштування ключів SSH.

Розумний HTTP напевно став найпопулярнішим методом використання Git, адже його можна налаштувати як для анонімної праці, як протокол git://, та до нього можна викладати з авторизацією та шифрування, як і з SSH протоколом. Замість налаштування різних URLів для цих речей, ви можете використовувати для них однин URL. Якщо ви спробуєте викласти, а сховище вимагає авторизації (як зазвичай і повинно бути), сервер запитає логін та пароль. Те ж саме стосується і доступу на читання.

Насправді, для таких сервісів як GitHub, URL, що ви використовуєте для перегляду сховища в мережі (наприклад, <https://github.com/schacon/simplegit>) збігається з URLом, що ви можете використовувати для клонування, та, якщо у вас є доступ, викладати до нього.

**Тупий HTTP**

Якщо сервер не відповідає на розумний сервіс HTTP, клієнт Git спробує відкотитись до простішого тупого HTTP протоколу. Тупий протокол очікує, що чисте сховище Git буде обслуговуватись як звичайні файли на веб сервері. Краса тупого протоколу HTTP в простоті його налаштування. Вам треба просто викласти чисте сховище Git під коренем документів HTTP та встановити потрібний гачок (хук, hook) post-update, ось і все (Дивіться [Гаки (hooks) Git](https://git-scm.com/book/uk/v2/ch00/_git_hooks)). Після цього, усі, в кого є доступ до веб серверу, на котрий ви скопіювали своє сховище, можуть зробити його клон.

1. Зберегти файл в кодуванні UTF-8