Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Информационных технологий и программирования

Работа: Лабораторная работа по Git 2

Выполнил: Гонтарь Тимур Сергеевич

Проверил: Повышев Владислав Вячеславович

Санкт-Петербург

2022 г.

Система контроля версий Git

Git — это распределенная система контроля версий. Иначе говоря, это набор консольных утилит, которые отслеживают и фиксируют изменения в файлах (чаще всего речь идет об исходном коде программ, но вы можно использовать его для любых файлов). Изначально Git был создан Линусом Торвальдсом при разработке ядра Linux.

К базовым возможностям Git относятся:

* возврат к любой предыдущей версии кода;
* просмотр истории изменений;
* параллельная работа над проектом;
* backup кода.

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и даёт возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом).

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранящие операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы. Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создаёт в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

Цикл работы в git:

* Редактирование, добавление, удаление файлов (собственно, работа).
* Индексация/добавление файлов в индекс (указание для git какие изменения нужно будет закоммитить).
* Коммит (фиксация изменений).
* Возврат к шагу 1 или отход ко сну.

Указатели:

* HEAD — указатель на текущий коммит или на текущую ветку (то есть, в любом

случае, на коммит). Указывает на родителя коммита, который будет создан

следующим.

* ORIG\_HEAD — указатель на коммит, с которого вы только что

переместили HEAD (командой git reset ..., например).

* Ветка (master, develop etc.) — указатель на коммит. При добавлении коммита,

указатель ветки перемещается с родительского коммита на новый.

* Теги — простые указатели на коммиты. Не перемещаются.

**Основные команды**

**Создание папок и файлов**

mkdir project # создать папку с именем «project»

touch index.html # создать файл

**Создать новый репозиторий**

git init # создать новый проект в текущей директории

git init folder-name # создать новый проект в указанной директории

**Клонирование репозитория**

# клонировать удаленный репозиторий в одноименную директорию

git clone https://github.com/cyberspacedk/Git-commands.git

# клонировать удаленный репозиторий в директорию «FolderName»

git clone https://github.com/cyberspacedk/Git-commands.git FolderName

# клонировать репозиторий в текущую директорию

git clone https://github.com:nicothin/web-design.git .

**Просмотр изменений**

git status # показать состояние репозитория (отслеживаемые, изменённые, новые

файлы и пр.)

git diff # сравнить рабочую директорию и индекс

**Добавление изменений в индекс**

git add . # добавить в индекс все новые, изменённые, удалённые файлы из текущей

директории и её поддиректорий

git add text.txt # добавить в индекс указанный файл (был изменён, был удалён или это

новый файл)

git add -p # показать новые/изменённые файлы по очереди с указанием их изменений и

вопросом об отслеживании/индексировании

**Удаление изменений из индекса**

git reset # убрать из индекса все добавленные в него изменения (в рабочей

директории все изменения сохранятся), антипод git add

git reset readme.txt # убрать из индекса изменения указанного файла (в рабочей директории

изменения сохранятся)

**Отмена изменений**

git checkout text.txt # ОПАСНО: отменить изменения в файле, вернуть состояние файла,

имеющееся в индексе

git reset --hard # ОПАСНО: отменить изменения; вернуть то, что в коммите, на

который указывает HEAD (незакомиченные изменения удалены из индекса и из рабочей

директории, неотслеживаемые файлы останутся на месте)

git clean -df # удалить неотслеживаемые файлы и директории

**Коммиты**

git commit -m "Name of commit" # зафиксировать в коммите проиндексированные

изменения (закоммитить), добавить сообщение

git commit -a -m "Name of commit" # проиндексировать отслеживаемые файлы (ТОЛЬКО

отслеживаемые, но НЕ новые файлы) и закоммитить, добавить сообщение

**Отмена коммитов и перемещение по истории**

git revert HEAD --no-edit # создать новый коммит, отменяющий изменения последнего

коммита без запуска редактора сообщения

git revert b9533bb --no-edit # то же, но отменяются изменения, внесённые коммитом с

указанным хешем (b9533bb)

**Временно переключиться на другой коммит**

git checkout b9533bb # переключиться на коммит с указанным хешем (переместить HEAD

на указанный коммит, рабочую директорию вернуть к состоянию, на момент этого

коммита)

git checkout master # переключиться на коммит, на который указывает master

(переместить HEAD на коммит, на который указывает master, рабочую директорию

вернуть к состоянию на момент этого коммита)

**Переключиться на другой коммит и продолжить работу с него**

git checkout -b new-branch 5589877 # создать ветку new-branch, начинающуюся с коммита

c хешем 5589877 (переместить HEAD на указанный коммит, рабочую директорию вернуть

к состоянию, на момент этого коммита, создать указатель на этот коммит (ветку) с

указанным именем)

**Восстановление изменений**

git checkout 5589877 index.html # восстановить в рабочей директории указанный файл на

момент указанного коммита (и добавить это изменение в индекс) (git reset index.html для

удаления из индекса, но сохранения изменений в файле)

**Копирование коммита (перенос коммитов)**

git cherry-pick 5589877 # скопировать на активную ветку изменения из указанного

коммита, закоммитить эти изменения

**Удаление файла**

git rm text.txt # удалить отслеживаемый неизменённый файл и проиндексировать это

изменение

git rm -f text.txt # удалить отслеживаемый изменённый файл и проиндексировать это

изменение

**Перемещение/переименование файлов**

git mv text.txt test\_new.txt # переименовать файл «text.txt» в «test\_new.txt» и

проиндексировать это изменение

git mv readme\_new.md folder/ # переместить файл readme\_new.md в директорию folder/

(должна существовать) и проиндексировать это изменение

**История коммитов**

git log master # показать коммиты в указанной ветке

git log -2 # показать последние 2 коммита в активной ветке

git log -2 --stat # показать последние 2 коммита и статистику внесенных ими

изменений

git log -p index.html # показать историю изменений файла index.html (коммиты и

изменения)

git log master..branch\_99 # показать коммиты из ветки branch\_99, которые не влиты в

master

git show 60d6582 # показать изменения из коммита с указанным хешем

**Ветки**

git branch # показать список веток

git branch -v # показать список веток и последний коммит в каждой

git branch new\_branch # создать новую ветку с указанным именем на текущем коммите

git branch new\_branch 5589877 # создать новую ветку с указанным именем на указанном

коммите

git branch -f master 5589877 # переместить ветку master на указанный коммит

git checkout new\_branch # перейти в указанную ветку

git checkout -b new\_branch # создать новую ветку с указанным именем и перейти в неё

git merge hotfix # влить в ветку, в которой находимся, данные из ветки hotfix

git branch --merged # показать ветки, уже слитые с активной

git branch --no-merged # показать ветки, не слитые с активной

git branch -a # показать все имеющиеся ветки (в т.ч. на удаленных репозиториях)

git branch -m old\_branch\_name new\_branch\_name # переименовать локально ветку

old\_branch\_name в new\_branch\_name

git push origin :old\_branch\_name new\_branch\_name # применить переименование в

удаленном репозитории

**Теги**

git tag v1.0.0 # создать тег с указанным именем на коммите, на который указывает HEAD

git tag -a -m 'В продакшен!' v1.0.1 master # создать тег с описанием на том коммите, на

который смотрит ветка master

git tag -d v1.0.0 # удалить тег с указанным именем(ами)

git tag -n # показать все теги, и по 1 строке сообщения коммитов, на которые

они указывают

**Временное сохранение изменений без коммита**

git stash # временно сохранить незакоммиченные изменения и убрать их из рабочей

директории

**Удалённые репозитории**

git remote -v # показать список удалённых репозиториев, связанных с локальным

git branch -a # показать все ветки(локальные и удаленные)

git remote rm origin # удалить привязку удалённого репозитория

git fetch origin # скачать все ветки с удаленного репозитория, но не сливать со

своими ветками

git push origin master # отправить в удалённый репозиторий (с сокр. именем origin)

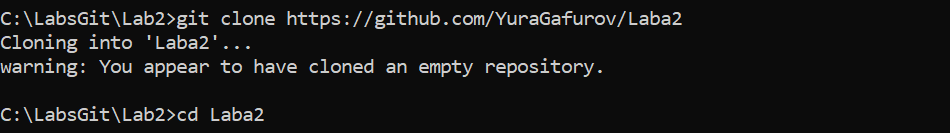
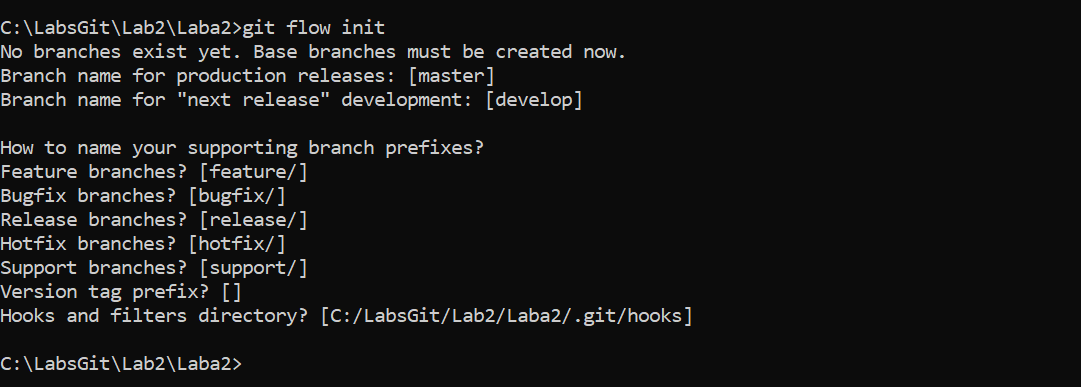
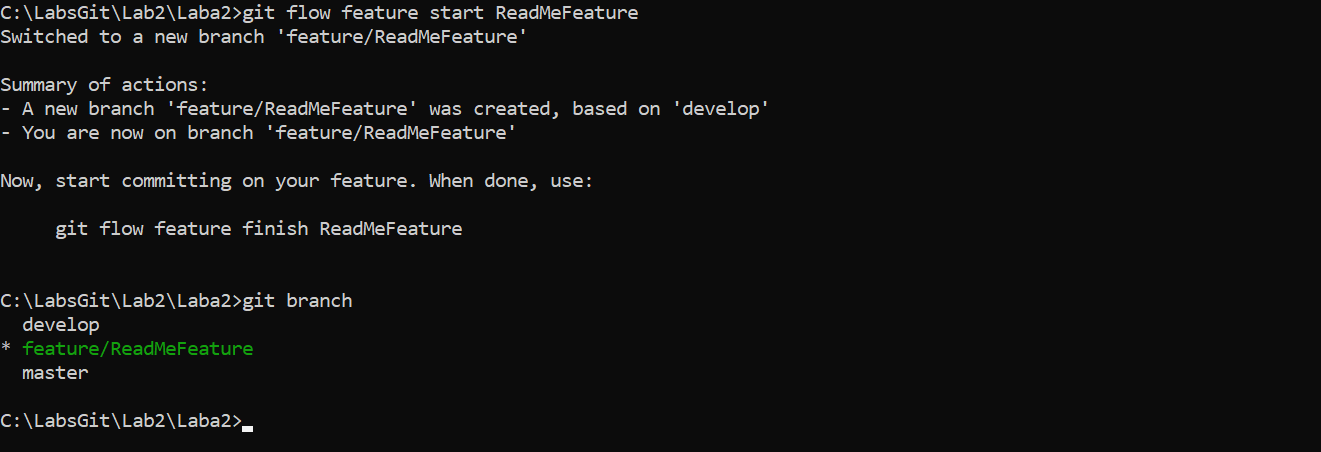
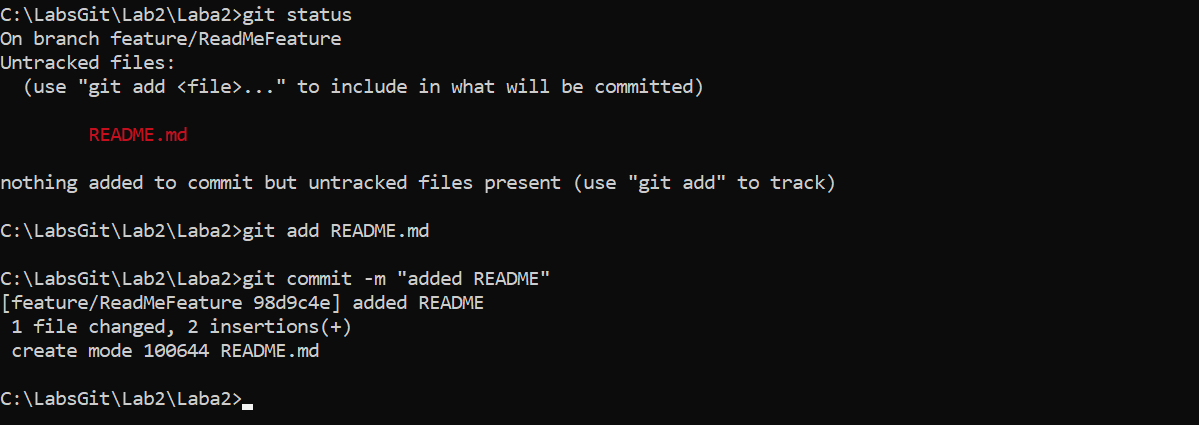
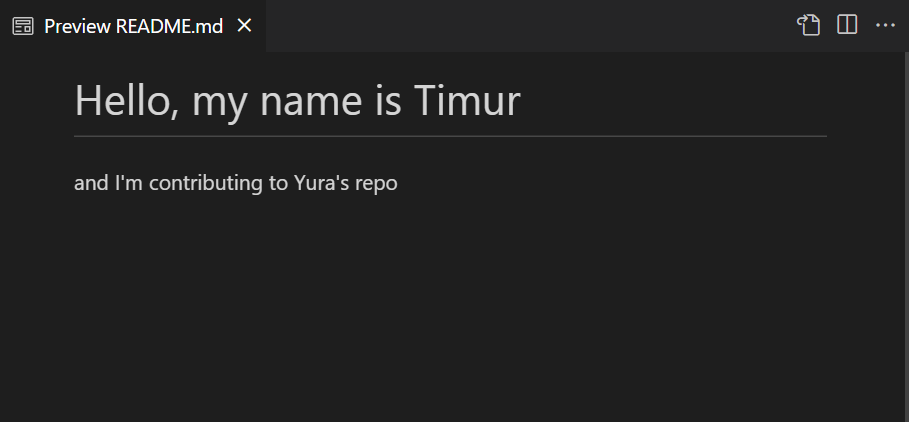
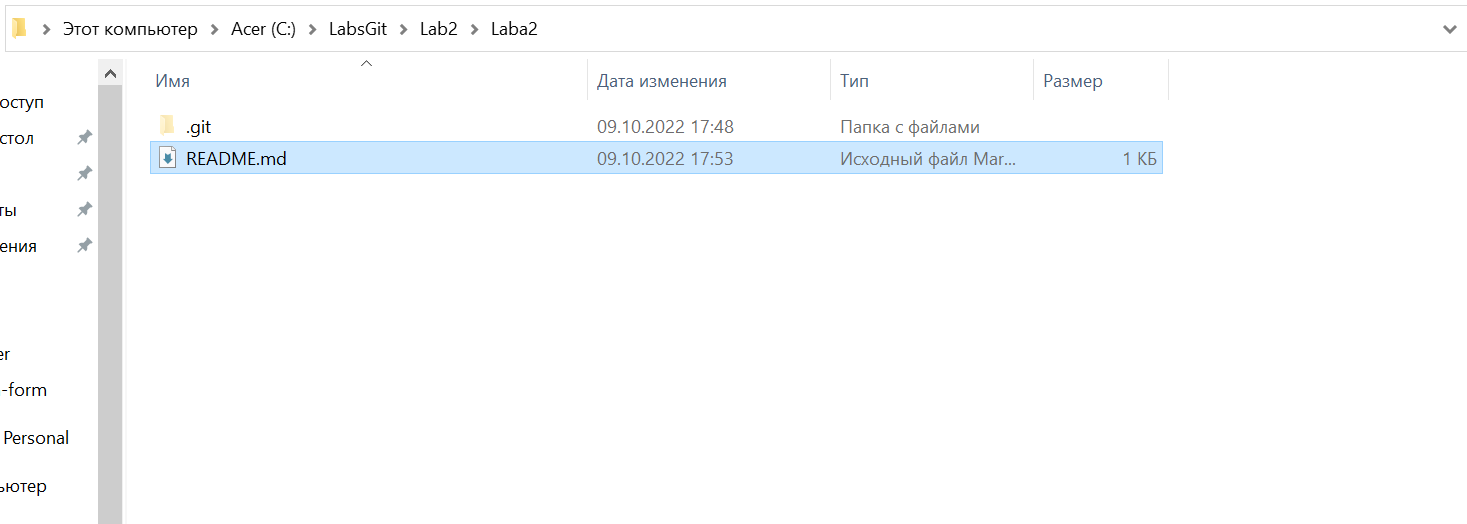
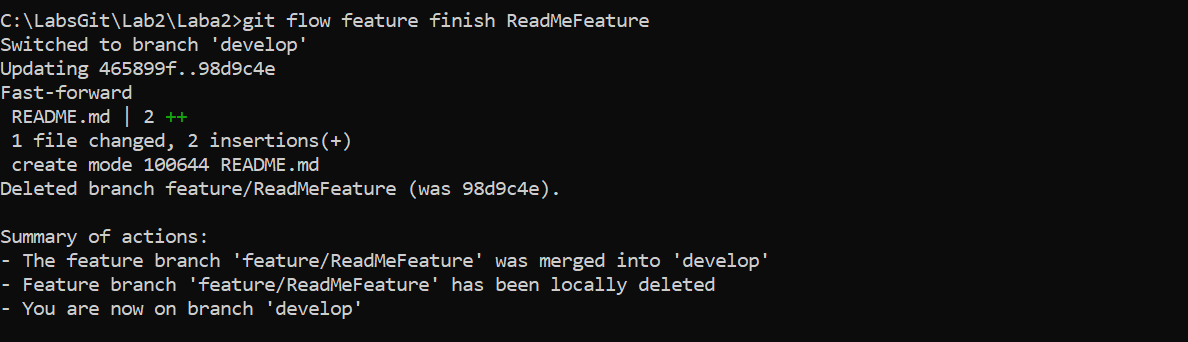
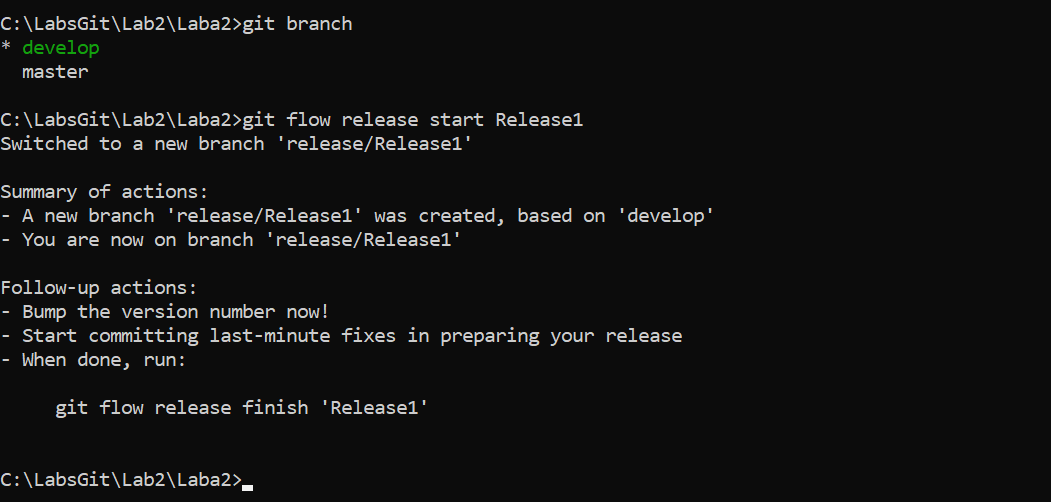
данные своей ветки master

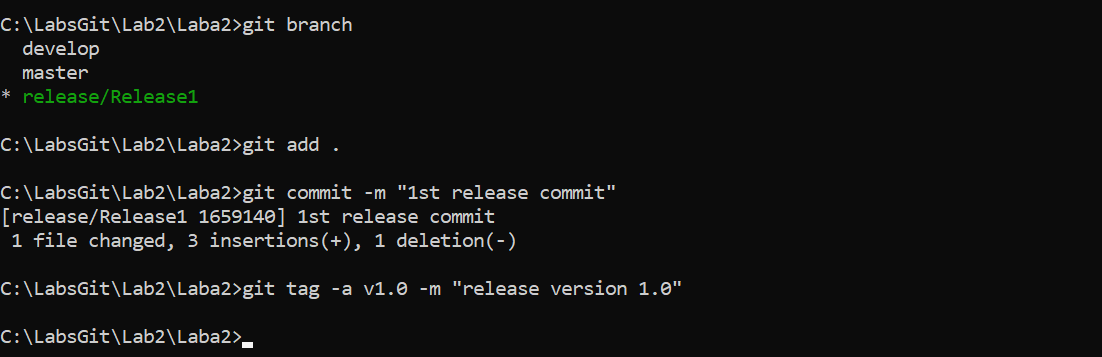
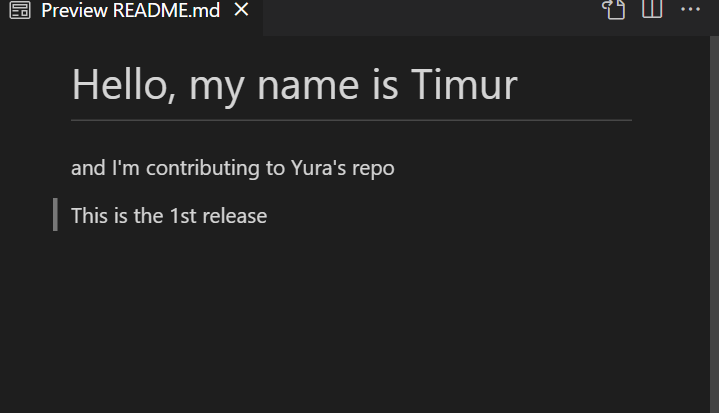
git pull origin master # влить изменения с удалённого репозитория (только указанная ветка)

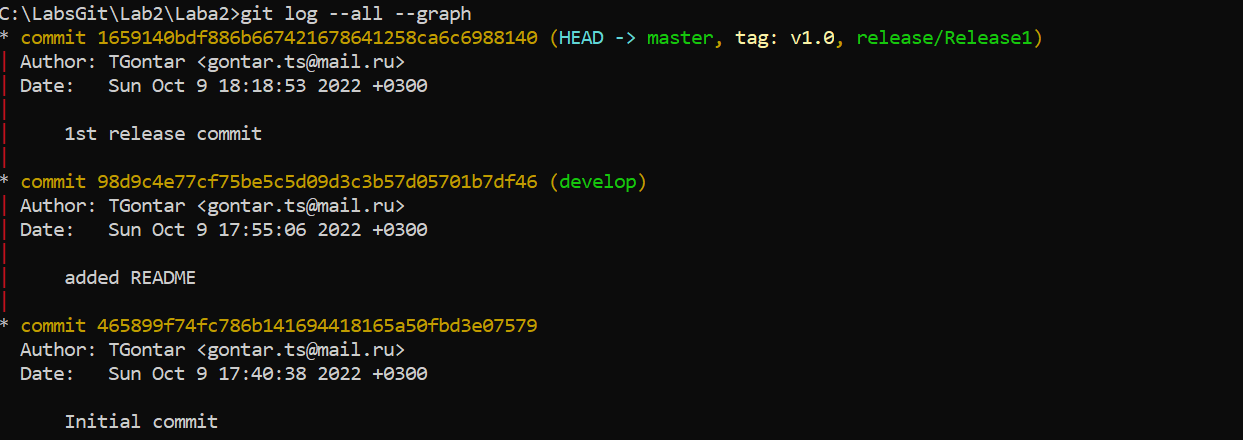
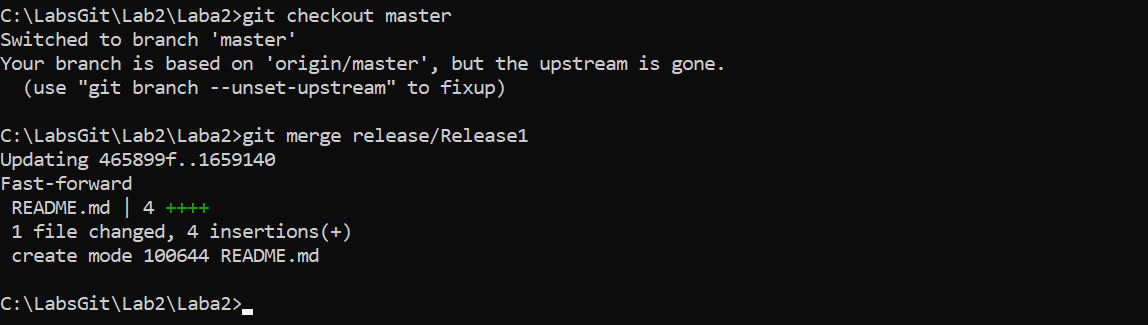
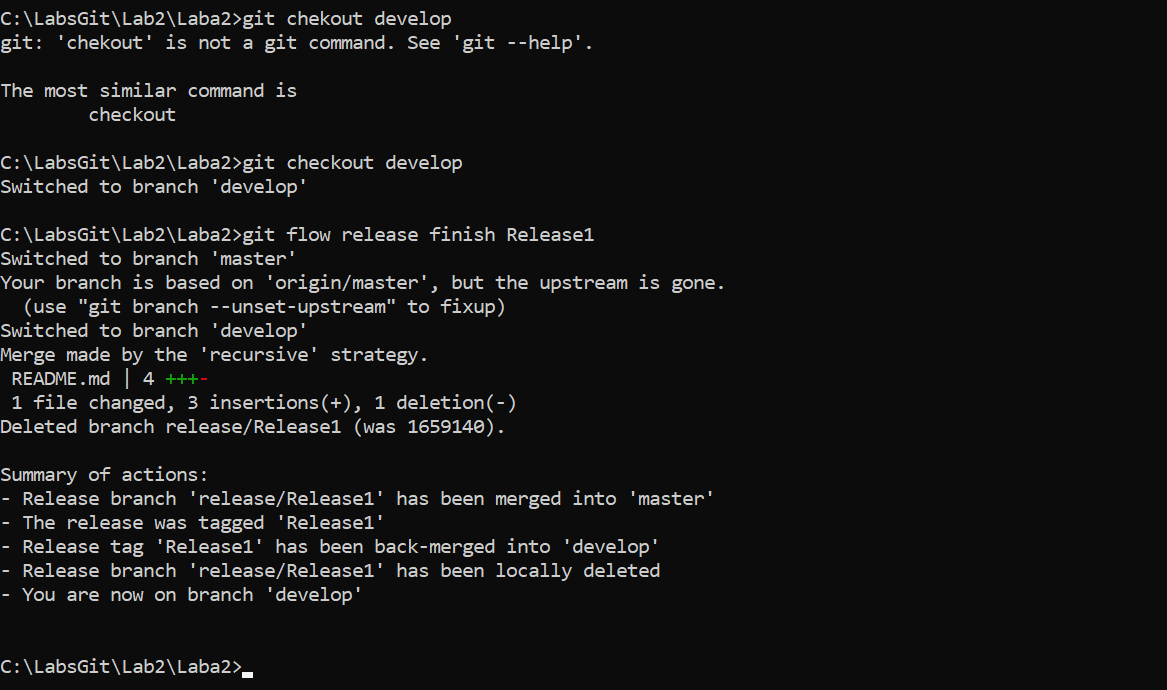
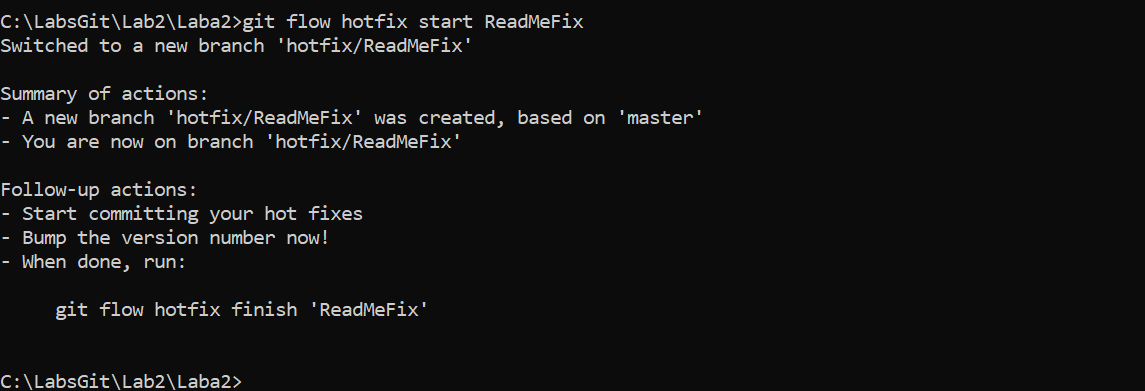
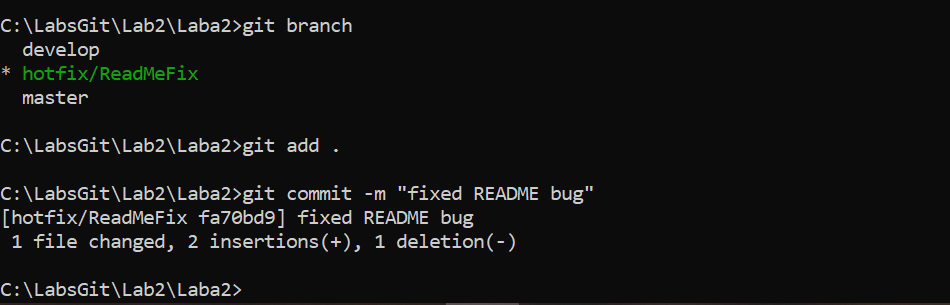
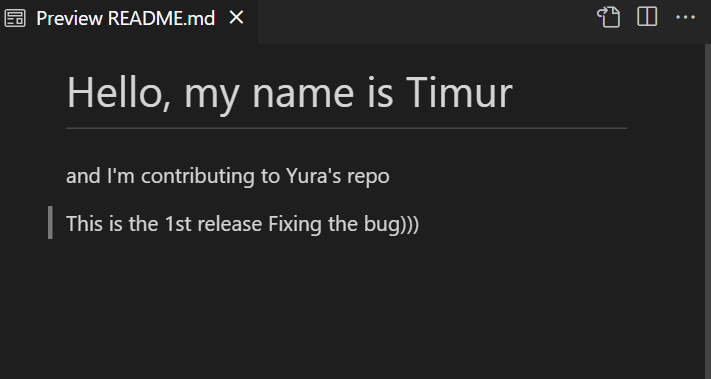
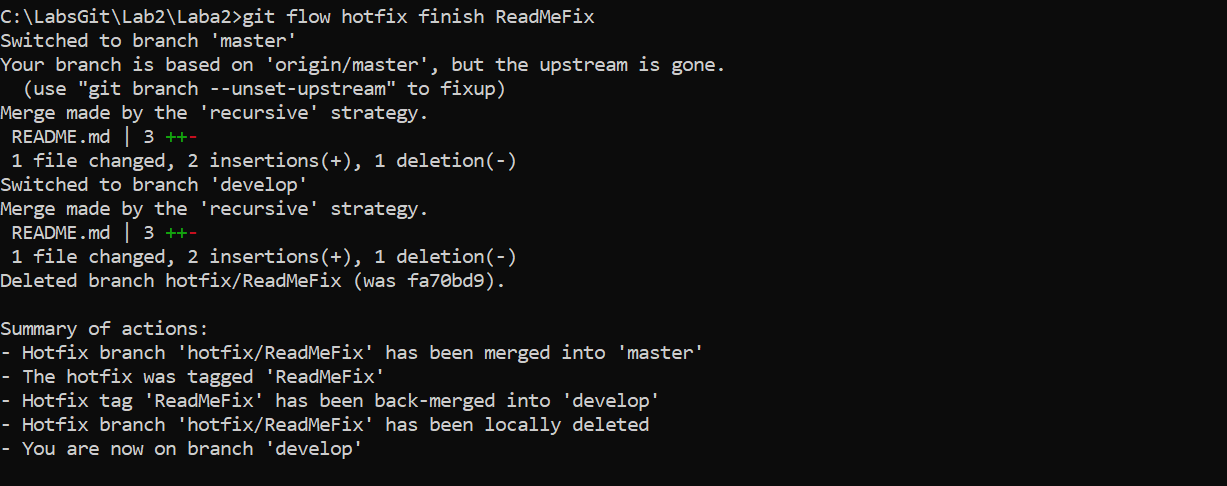
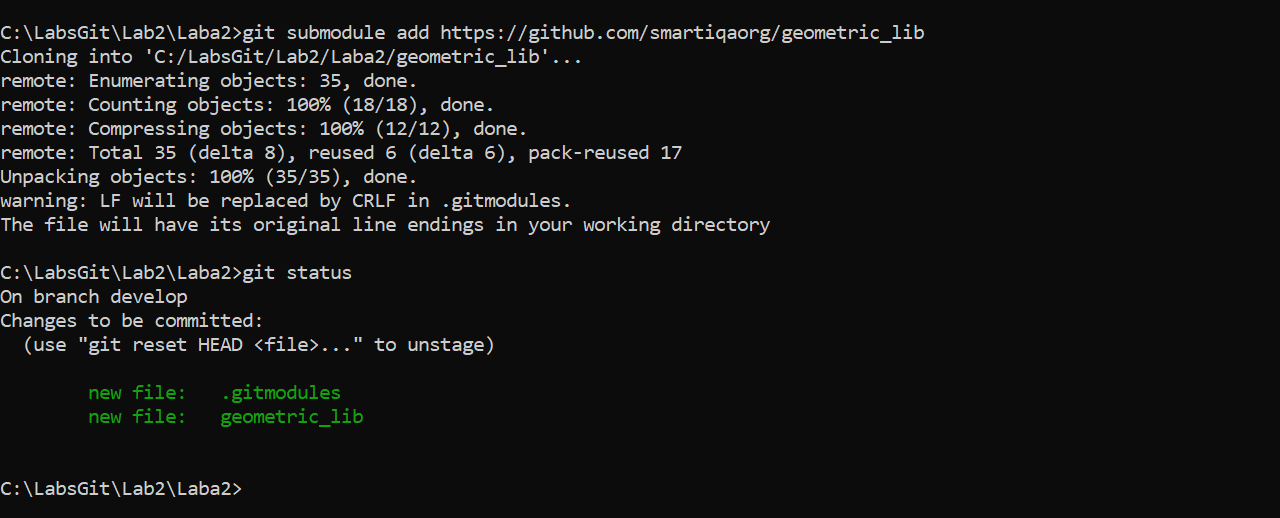
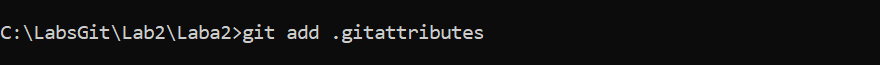
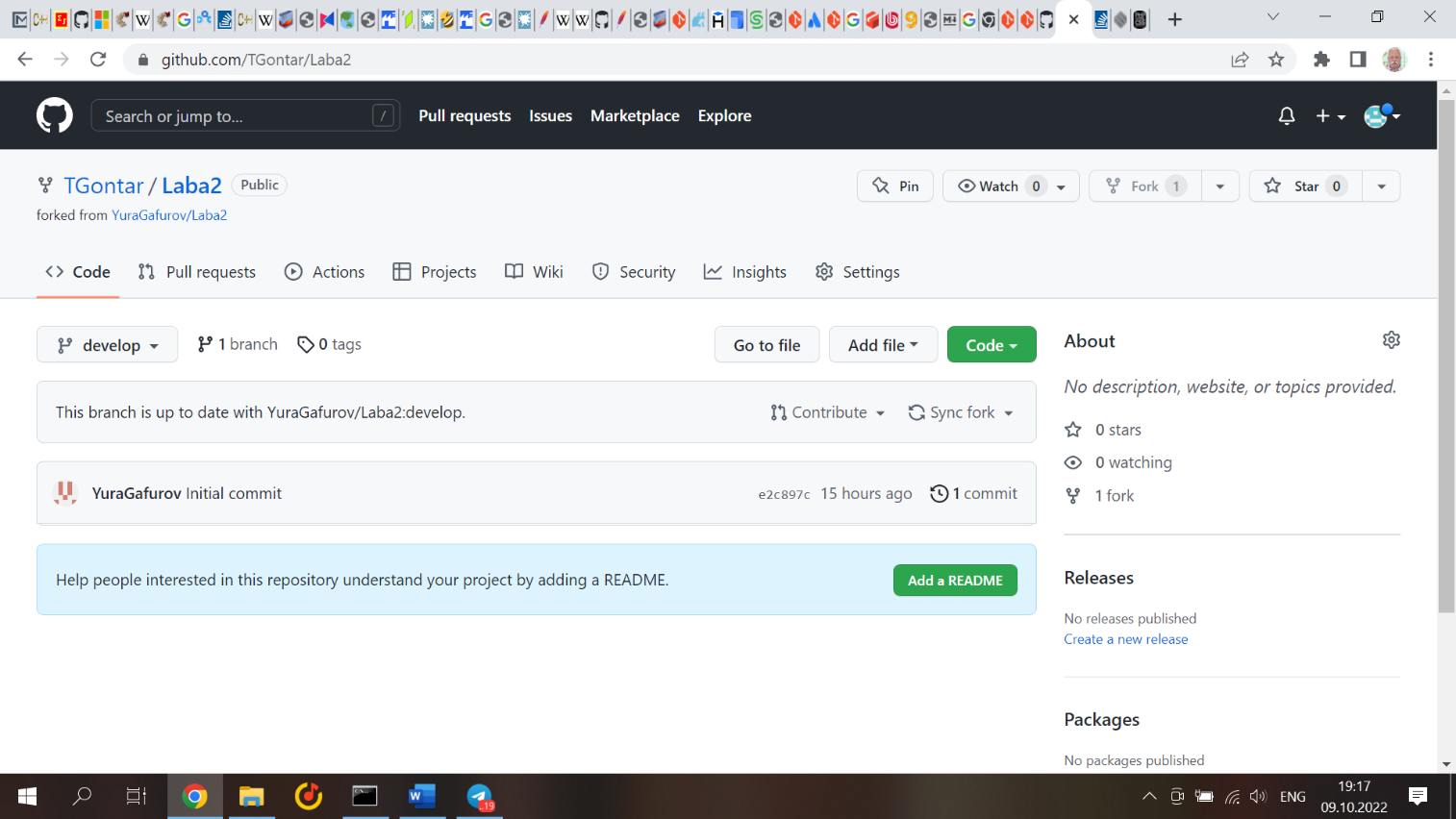
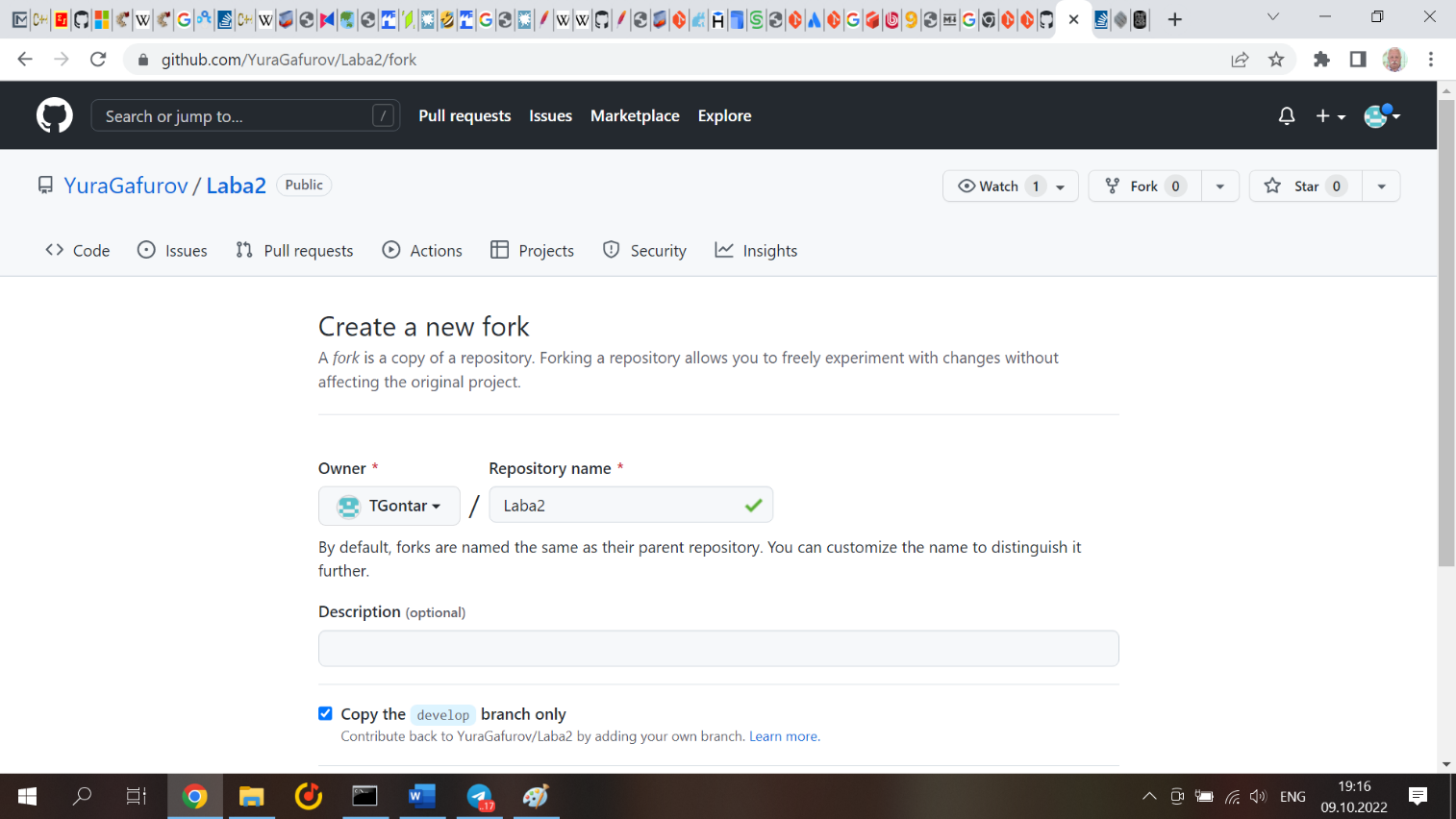
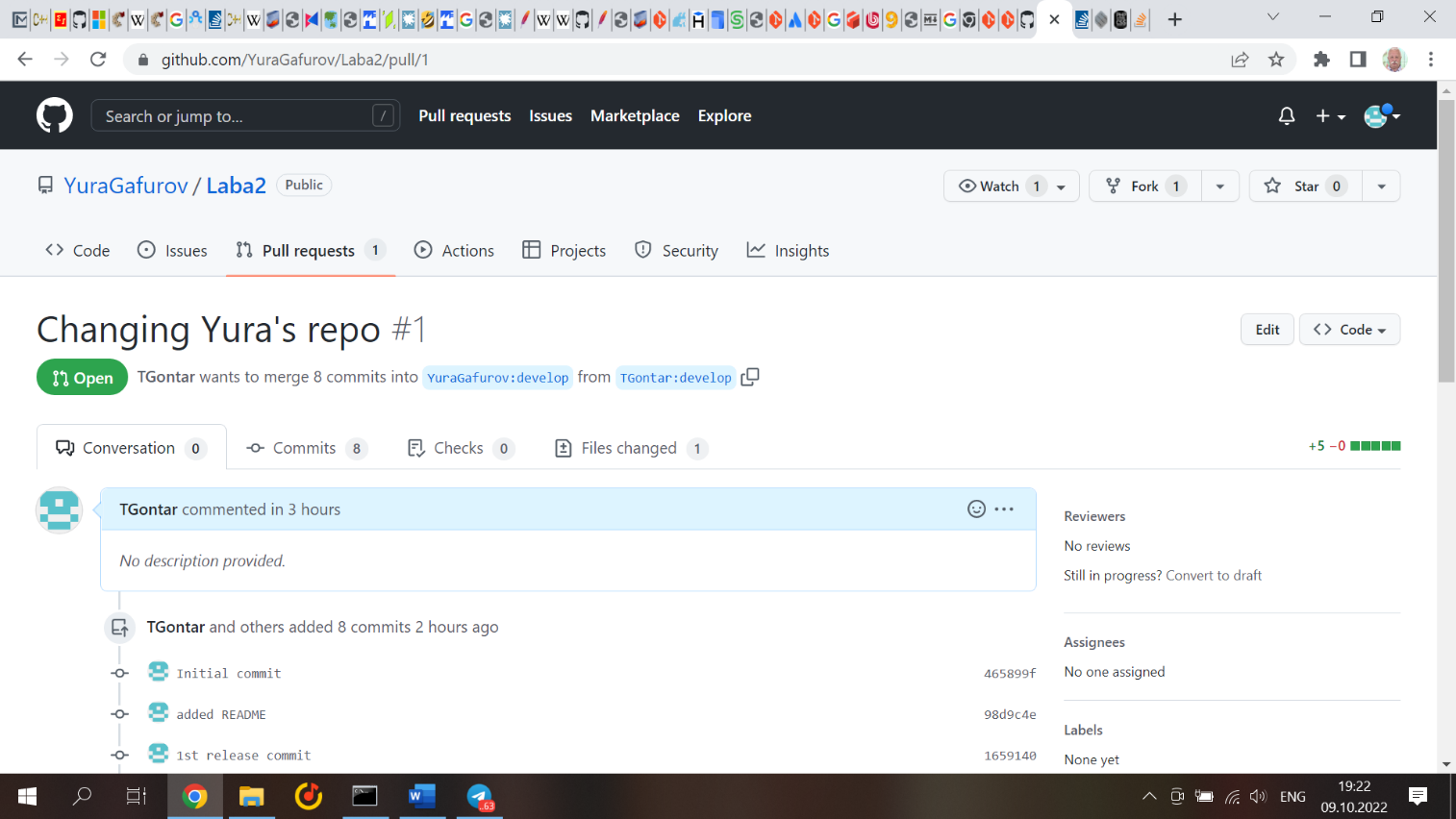
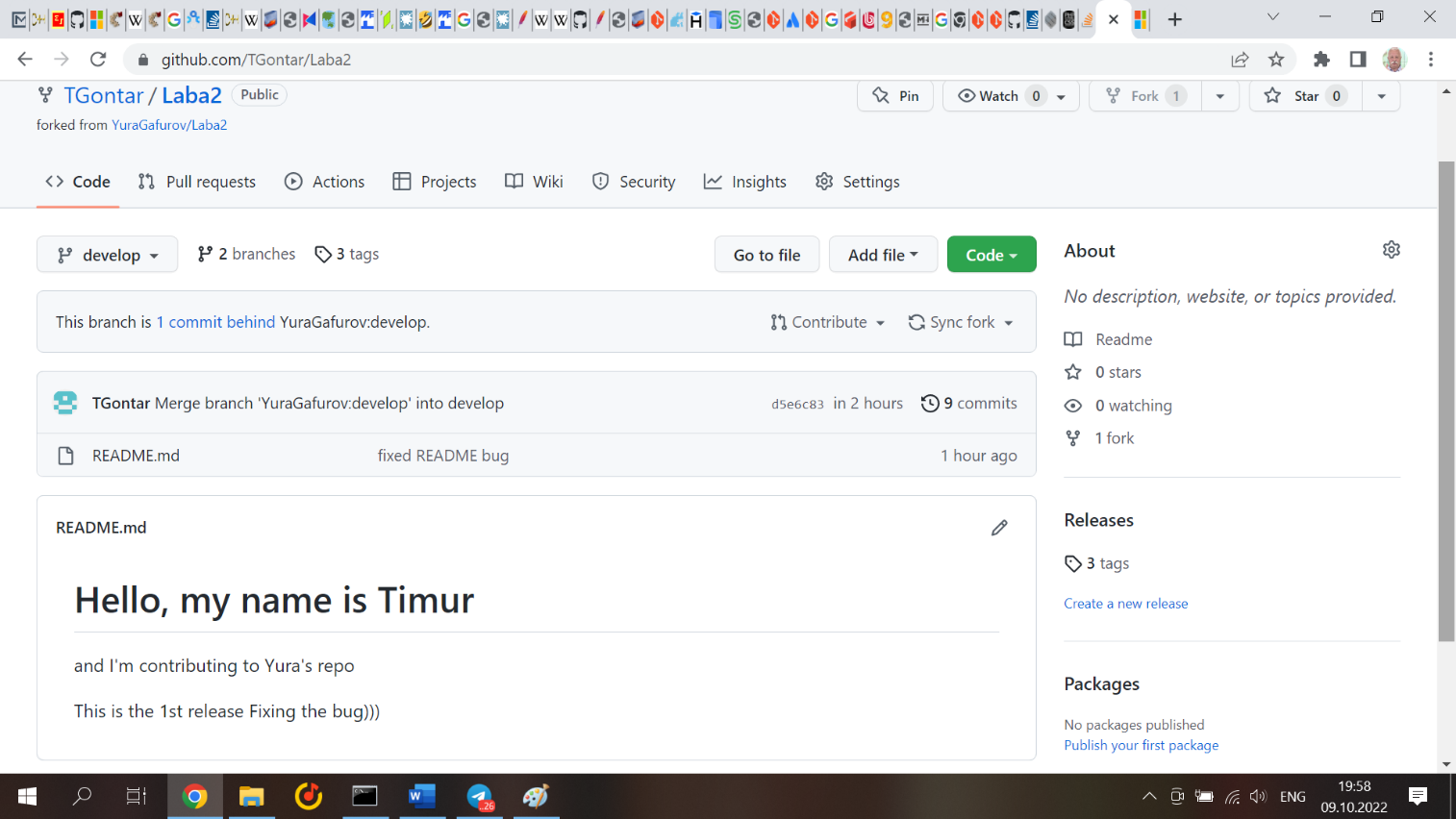
**Конфликт слияния**

git merge feature # влить в активную ветку изменения из ветки feature

**GIT FLOW**

1. Клонирую репозиторий партнера для работы с ним и последующего pull requesta
2. Инициализирую gitflow, создаётся ветка develop
3. Cоздаю ветку ReadMeFeature, туда я добавлю файл ReadMe
4. Создаю файл ReadMe, добавляю его в репозиторий и делаю commit
5. Завершаю работу с веткой feature – ветка сливается в develop и после этого удаляется
6. Создаю ветку Release1, где будет релиз версии 1.0 нашего проекта
7. Изменяю файл ReadMe и коммичу это. Также ставлю тег на этот коммит, сообщающий что это версия 1.0



1. Сливаю ветку релиза и мастер вручную
2. Завершаю работу с веткой релиз, результат сливается в develop
3. Запускаю ветку hotfix 
4. Изменяю файл ReadMe для ветки хотфикс и коммичу это
5. Завершаю работу с хотфиксом, результат сливается в master и develop
6. Добавляю подмодуль (submodule), им будет выступать репозиторий из предыдущей лабораторной
7. Ставлю LFS – расширение будет отслеживать и хранить в отдельном хранилище все файлы типа md
8. Делаю fork проекта партнера, чтобы залить создать pull request
9. Пушу все изменения в этот fork и создаю pull request
10. Партнёр принимает pull request. Работа с репозиторием окончена