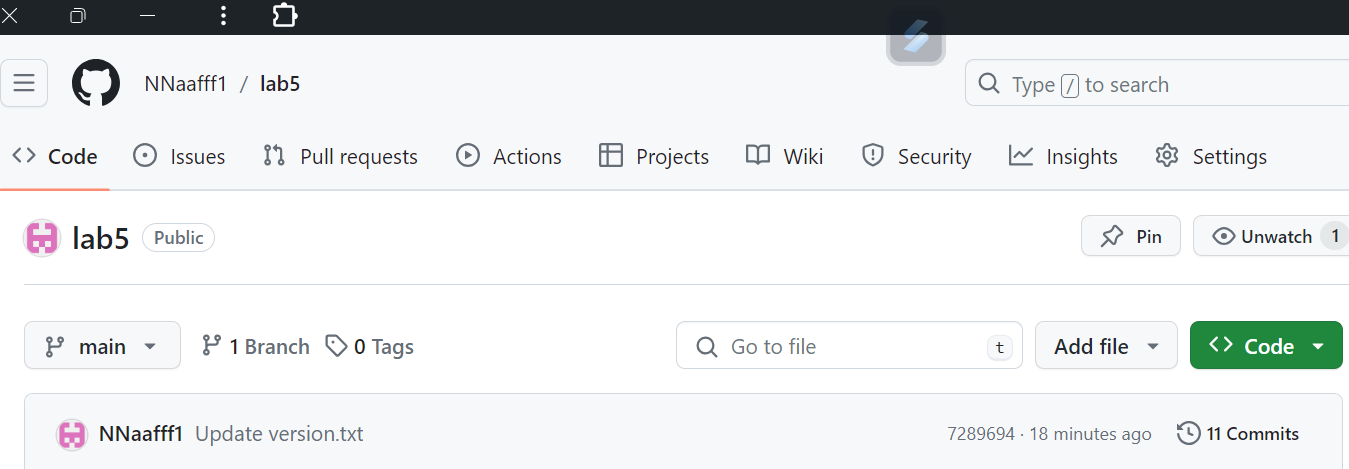
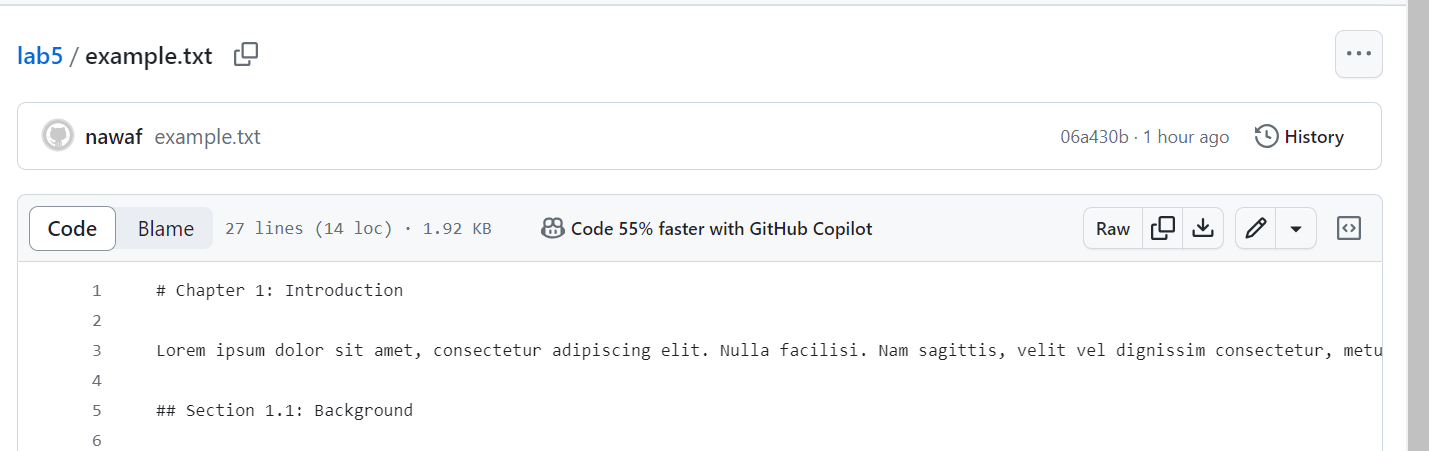
**Отчет по лабораторной работе №5**

## Введение

1. Сначала я создал репозиторий git-practice на GitHub и скопировал его URL-адрес.

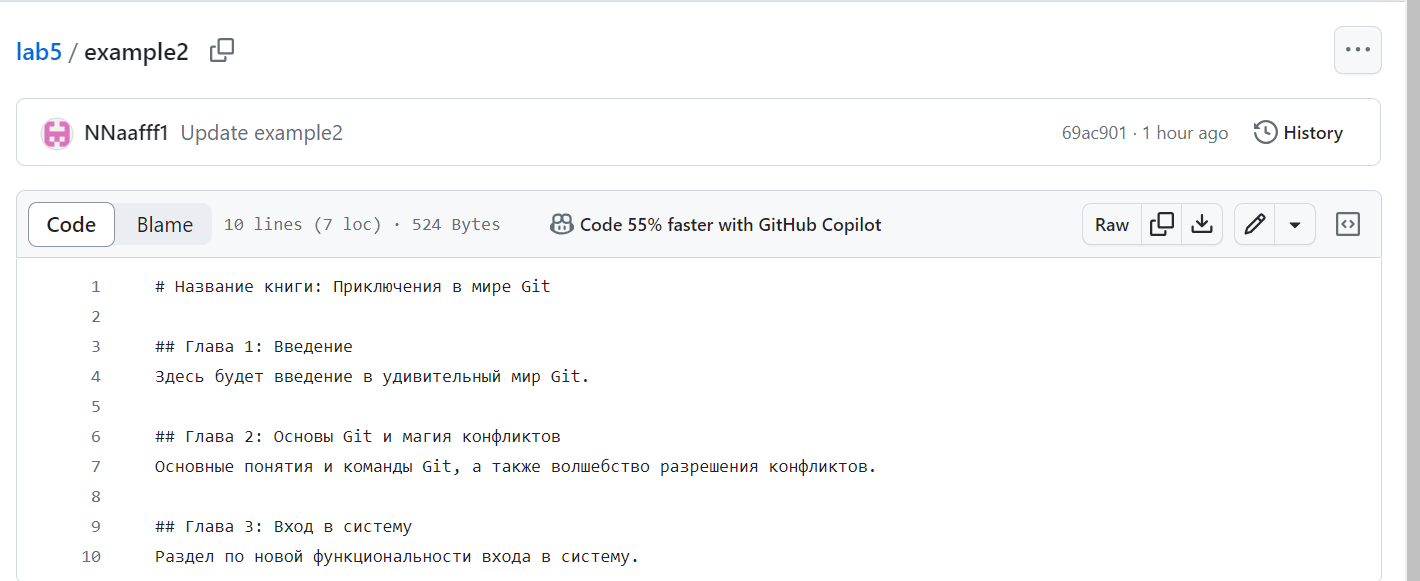
2.

1. Далее с помощью терминала и команд *git clone* и *cd* я перешел в папку для локального сохранения репозитория.
2. Потом я создал новый текстовый файл example.txt, добавил в него некоторый текст и запушил на GitHub в основуную ветку master, используя команды git add, git commit -m и git push.
3. Затем я создал новую ветку second branch с помощью команды git branch, переключился на нее с помощью git checkout.
4. После я отредактировал файл example.txt, повторил некоторые шаги из пункта 3, переключился обратно в основную ветку master помощью git checkout main и слил изменения из second branch в основную ветку, используя git merge и git push.

7.

## Работа с ветками

1. Я создал новый текстовый файл с базовой структурой книги kniga.txt.

 2.

3.С помощью git checkout -b создал новую ветку feature-login.

1. Далее я внес изменения в файл, закоммитил их и отправил ветку на GitHub, используя git add, git commit -m, git push origin.

## Работа с удаленным репозиторием

1. Я переключился на основную ветку *master*, внес изменения в файл *kniga.txt*, закоммитил изменения и отправил их на *GitHub* с помощью *git add*, *git commit -m*, *git push origin*.

## Моделирование конфликта

1. Я вернулся в ветку *feature-login*, изменил главу 2 в файле *kniga.txt*, закоммитил изменения и отправил их на *GitHub*, используя *git checkout*, *git add*, *git commit -m*, *git push origin*.

## Разрешение конфликта

1. Я вернулся в основную ветку *master* и попробовал слить изменения с помощью *git checkout master* и *git pull origin master*. Возник конфликт.
2. Я разрешил конфликт, удалив метки и оставив нужные изменения.
3. Я закоммитил решение конфликта и отправил его на *GitHub* с помощью *git add*, *git commit -m*, *git push*.

## Автоматизация проверки формата файлов при коммите

1. Я создал bash-script, который будет выполнять проверку формата .txt файлов.



2.

1. Я добавил скрипт в репозиторий, поместив его в папку .git/hooks и убедившись, что у него есть права на выполнение с помощью cp и chmod +x.
2. Далее я попробовал внести изменения и закоммитить. Теперь, при каждой попытке закоммитить изменения, Git будет автоматически выполнять проверку формата файлов перед коммитом.
3. При возникновении необходимости внести изменения в файлы, чтобы они соответствовали формату, нужно внести изменения, добавить файлы и снова попробовать закоммитить.

## Использование Git Flow в проекте

1. Сначала я проверил, есть ли у меня на компьютере Git Flow.
2. Затем я выполнил инициализацию Git Flow в корне репозитория с помощью git flow init.
3. Создал ветку для новой функциональности "task-management" с помощью git flow feature start task-management.
4. Внес изменения в код для добавления функционала управления задачами task\_manager.py и выполнила коммит.

 5.

6.Потом я завершил фичу и объединил ее с основной веткой с помощью *git flow feature finish task-management*.

1. Далее я переключился на ветку *develop* и начал создание релиза:

git checkout develop

git flow release start v1.0.0

1. Я внес изменения, связанные с релизом:

echo "v1.0.0" > version.txt

git add version.txt

git commit -m "Обновлена версия для релиза v1.0.0"

1. Я завершил релиз и объединил его с ветками *develop* и *master*:

git flow release finish v1.0.0

1. Создал *hotfix*, если в процессе использования выявлена критическая ошибка:

git flow hotfix start hotfix-1.0.1

1. Внес изменения для исправления ошибки и закоммитил:

git add file\_with\_error.py

git commit -m "Исправлена критическая ошибка"

1. Завершил *hotfix* и объедините его с ветками *develop* и *master*:

git flow hotfix finish hotfix-1.0.1

1. Отправил изменения на удаленный репозиторий:

git push origin develop

git push origin main