**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

*дисциплина: Операционные системы*

Студент: Манаева Варвара Евгеньевна

Группа: НФИбд-01-20

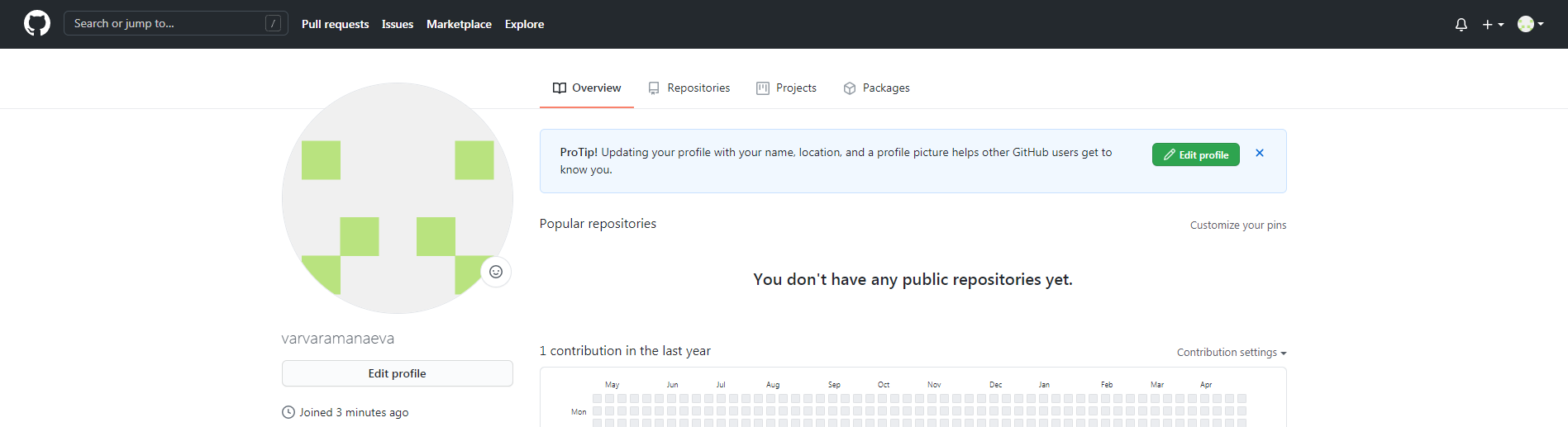
**МОСКВА**

2021 г.

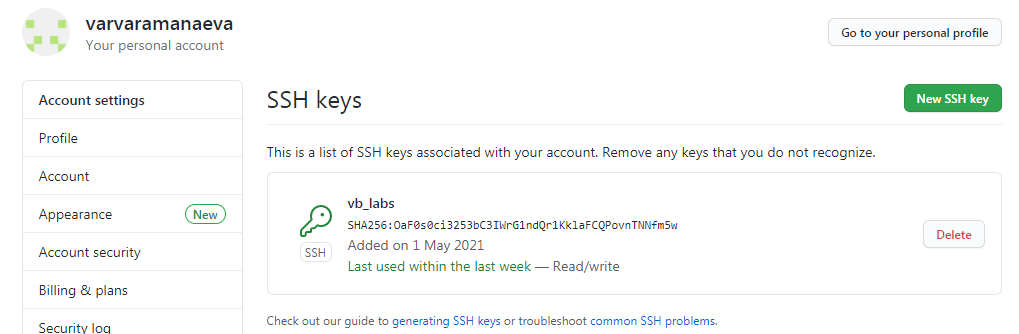
**Цель:** изучение идеологии и применения средств контроля версий.

**Этапы работы:**

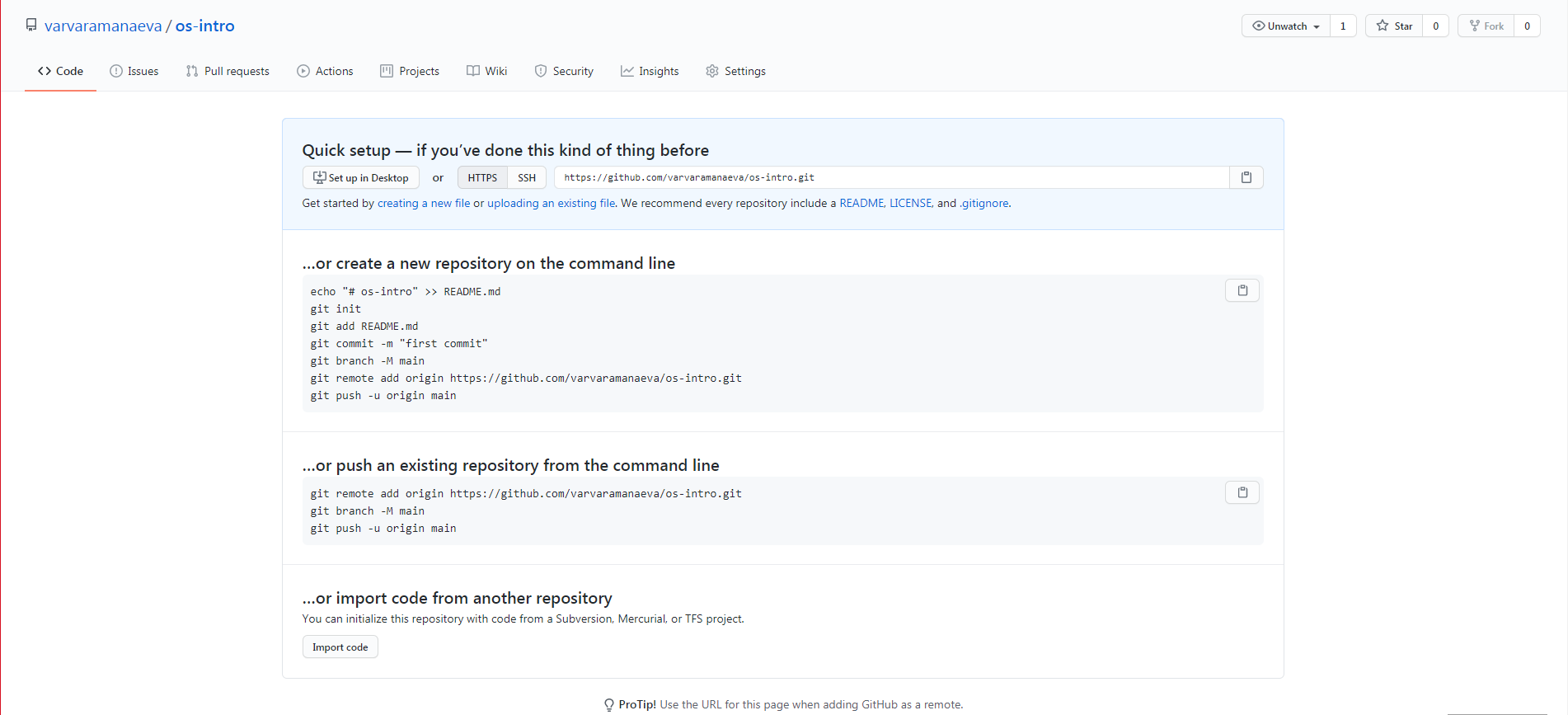
1. Была создана учётная запись на *github.com*.



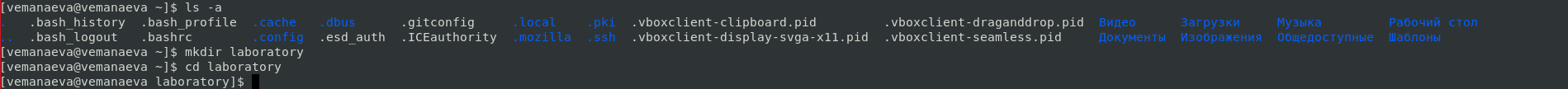
1. Была настроена система контроля версий *git* для репозиториев *github.com*.



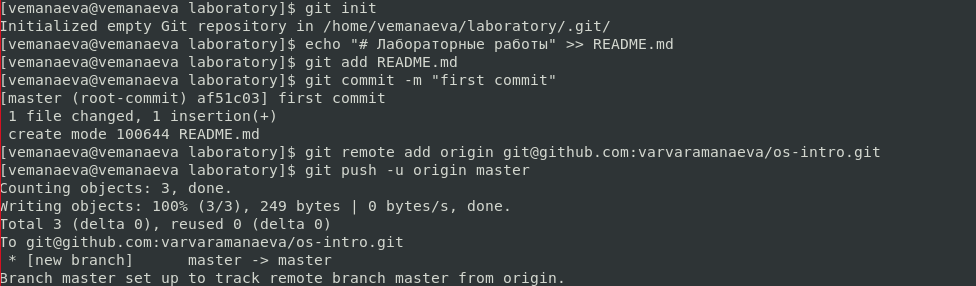
1. Был создан репозиторий на *github.com* под названием *os-intro*.



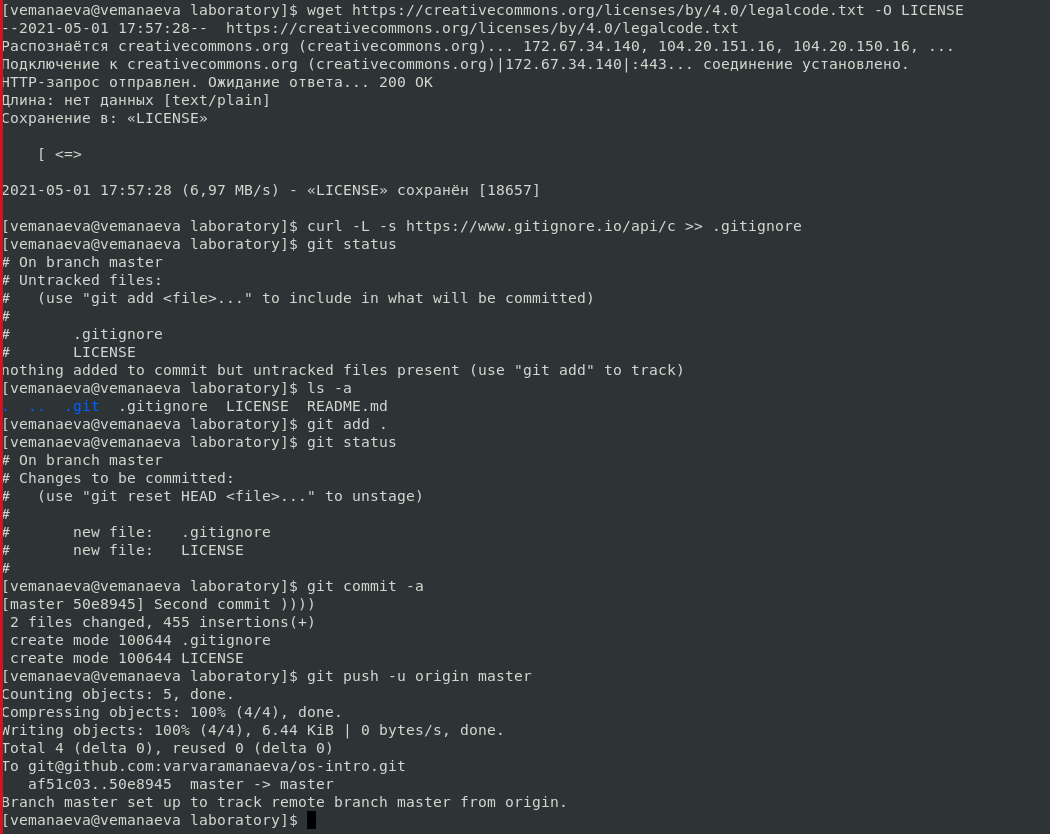
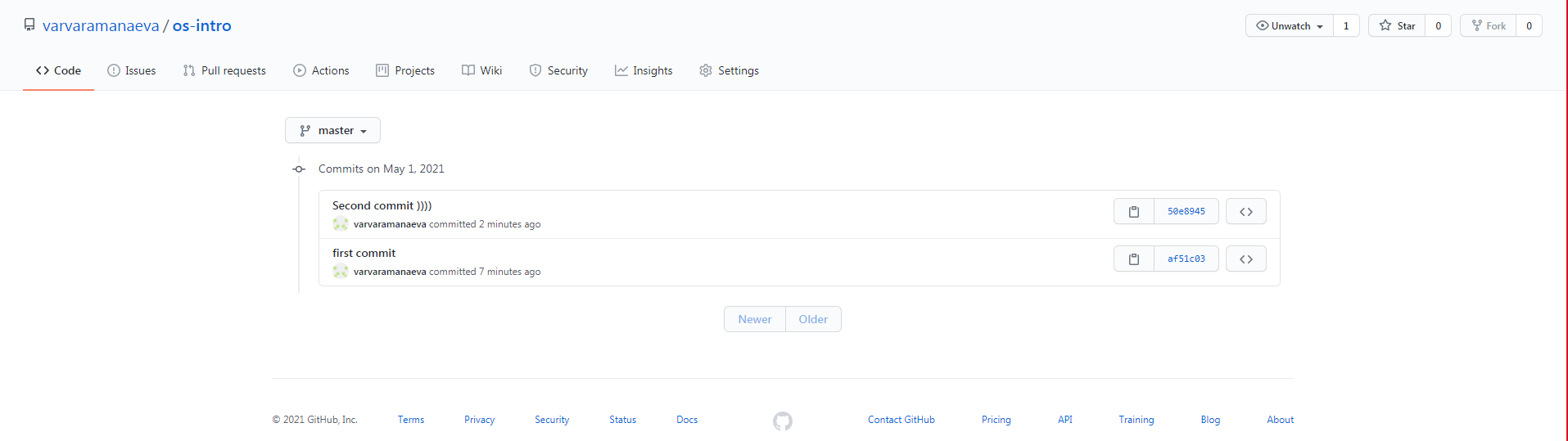
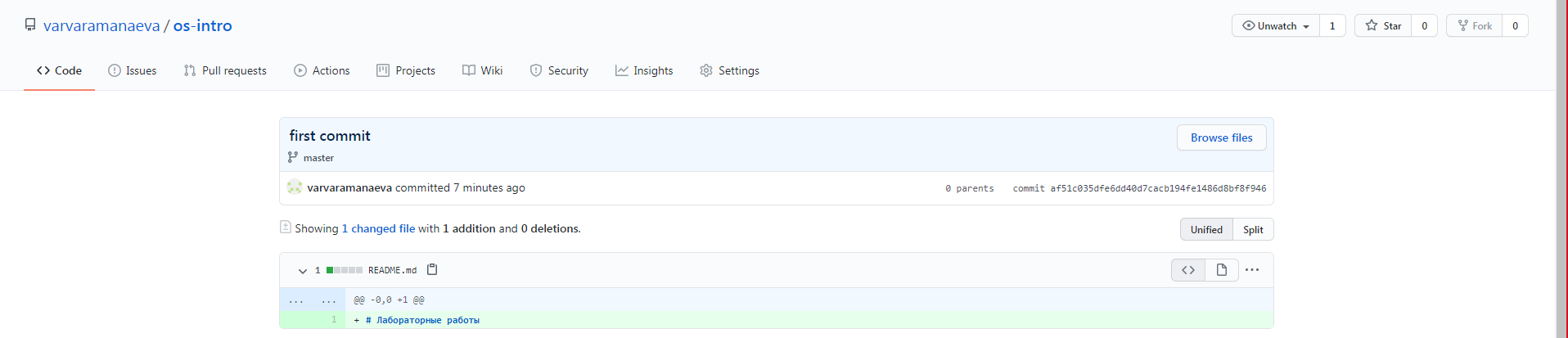
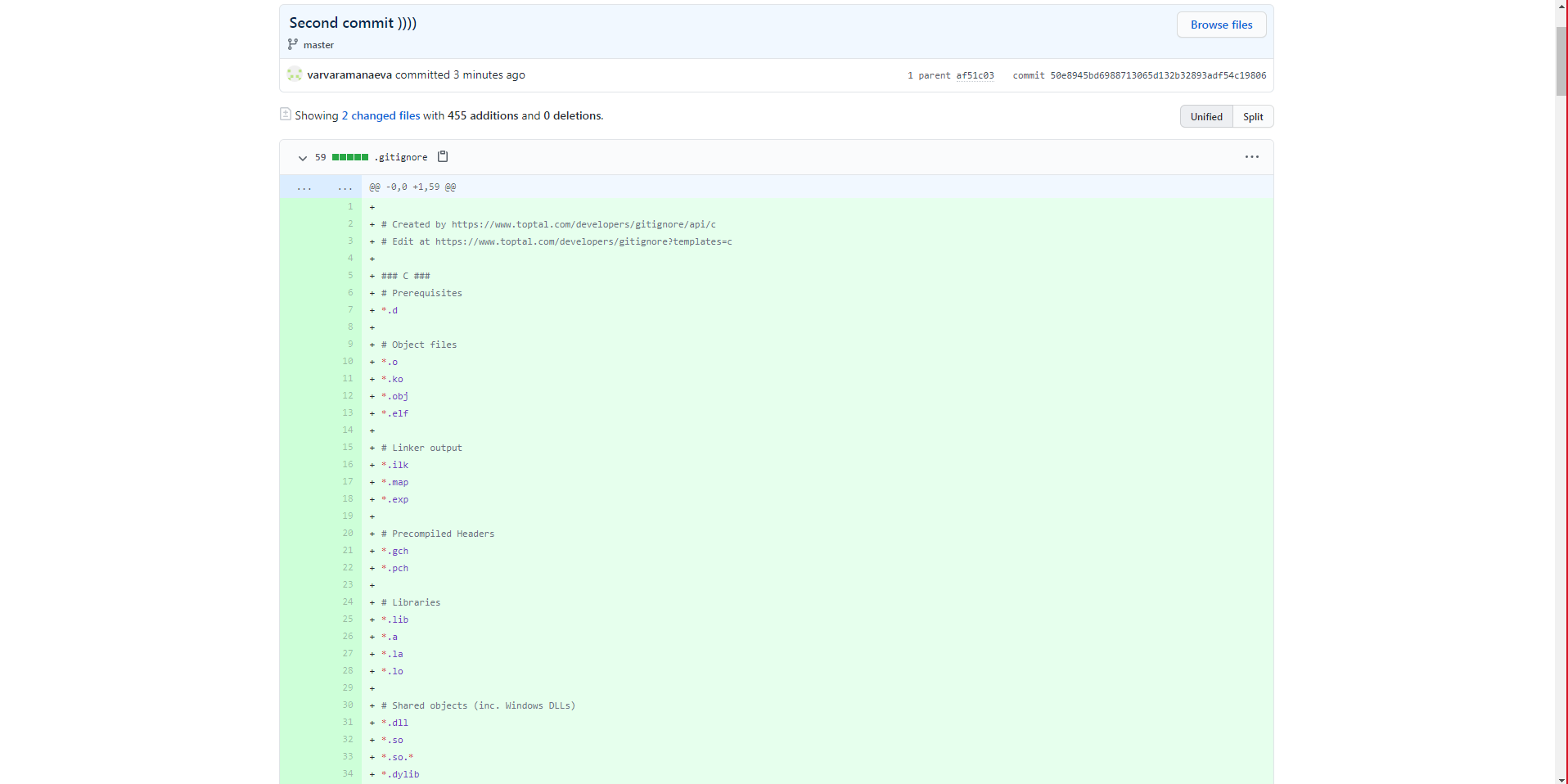
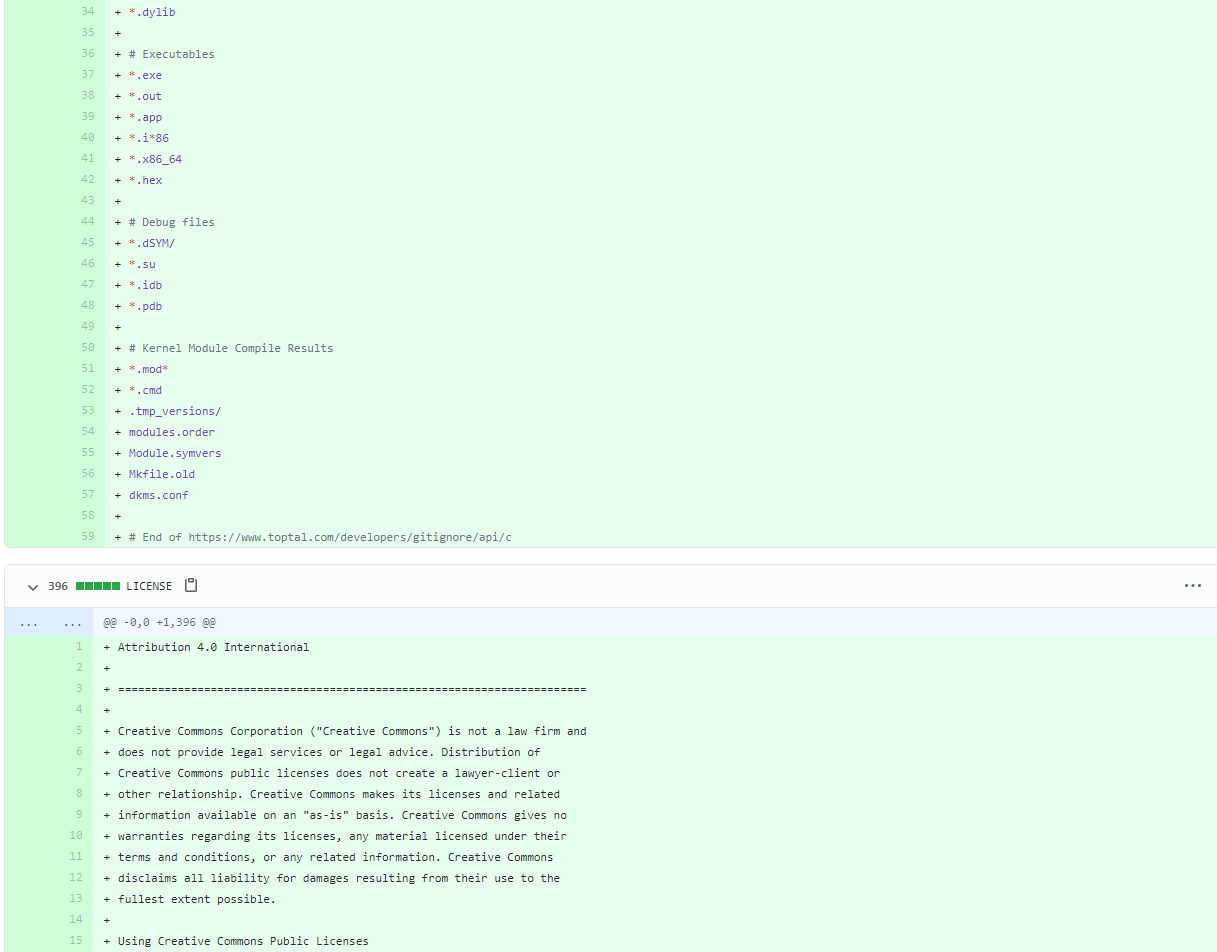
1. Создан рабочий каталог.



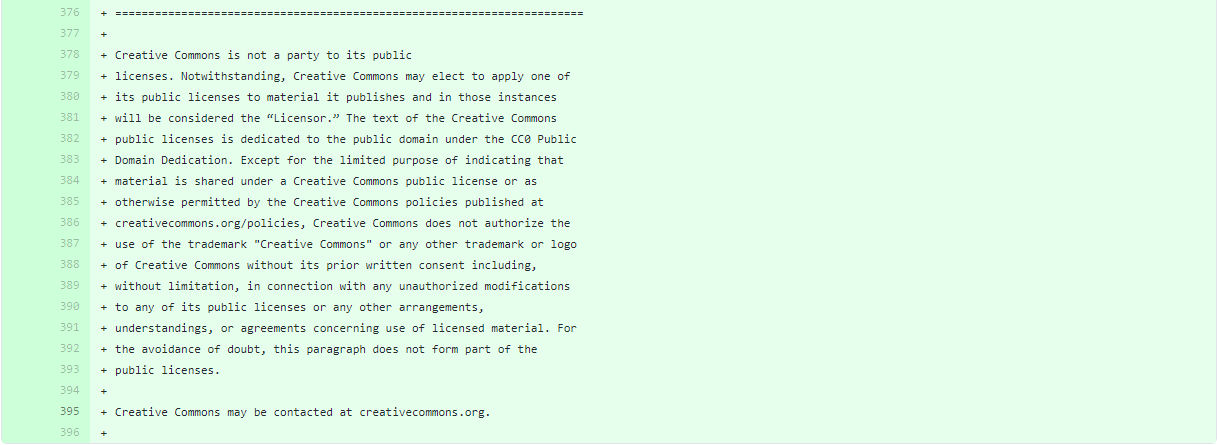
1. Рабочий каталог был подключен к системе *git*. В нём был создан файл *README.md*, с записью «Лабораторные работы». Файл отправлен на *github.com*.



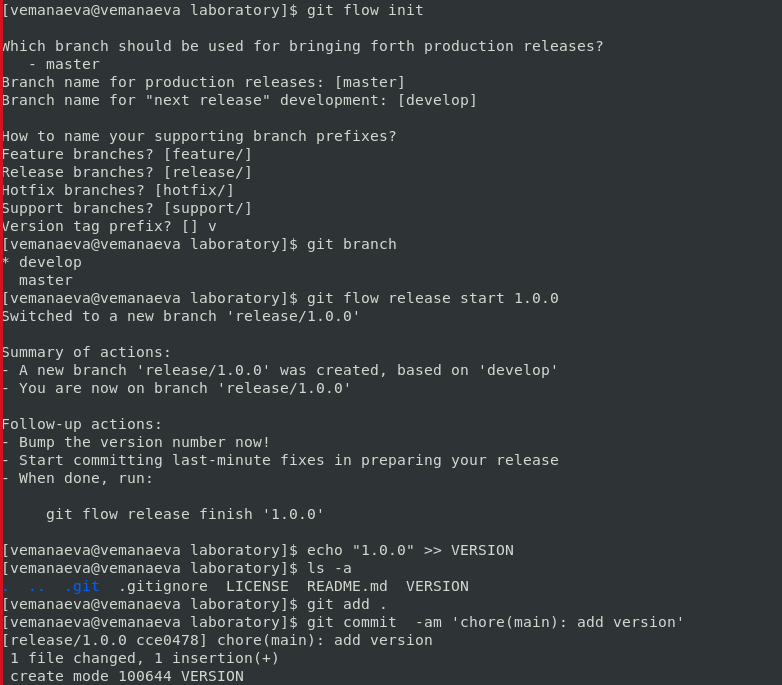
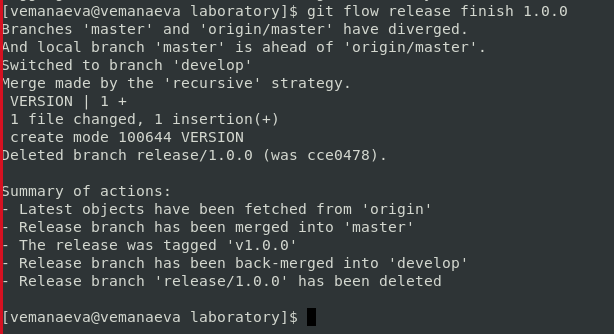
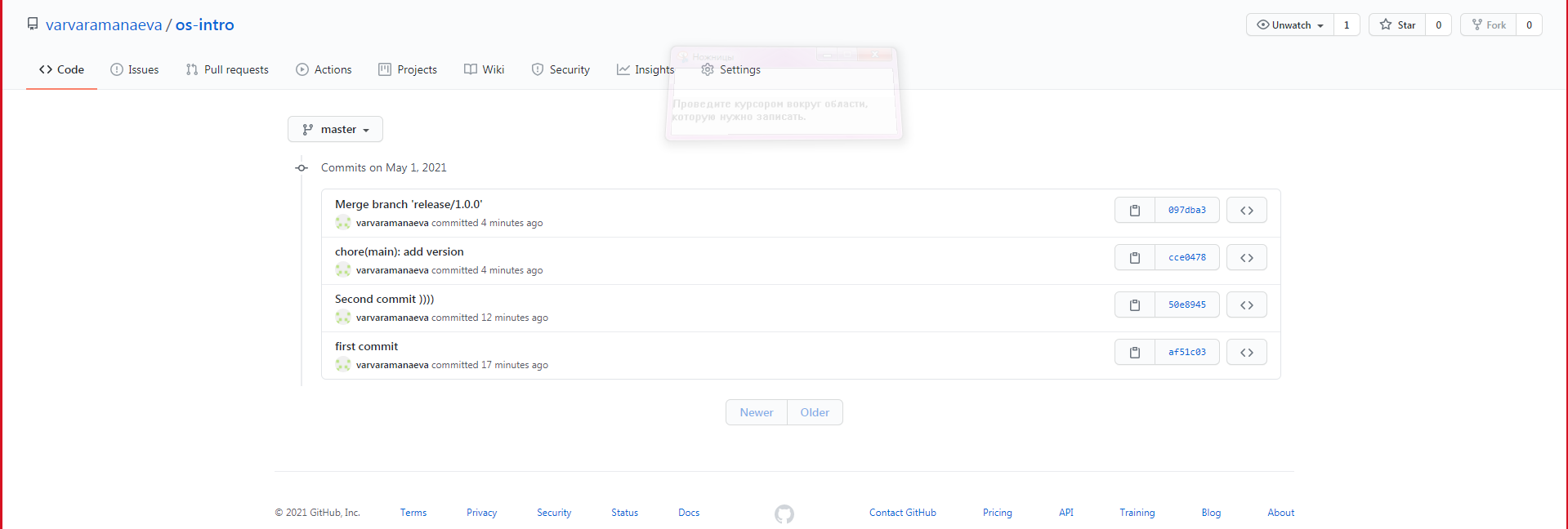
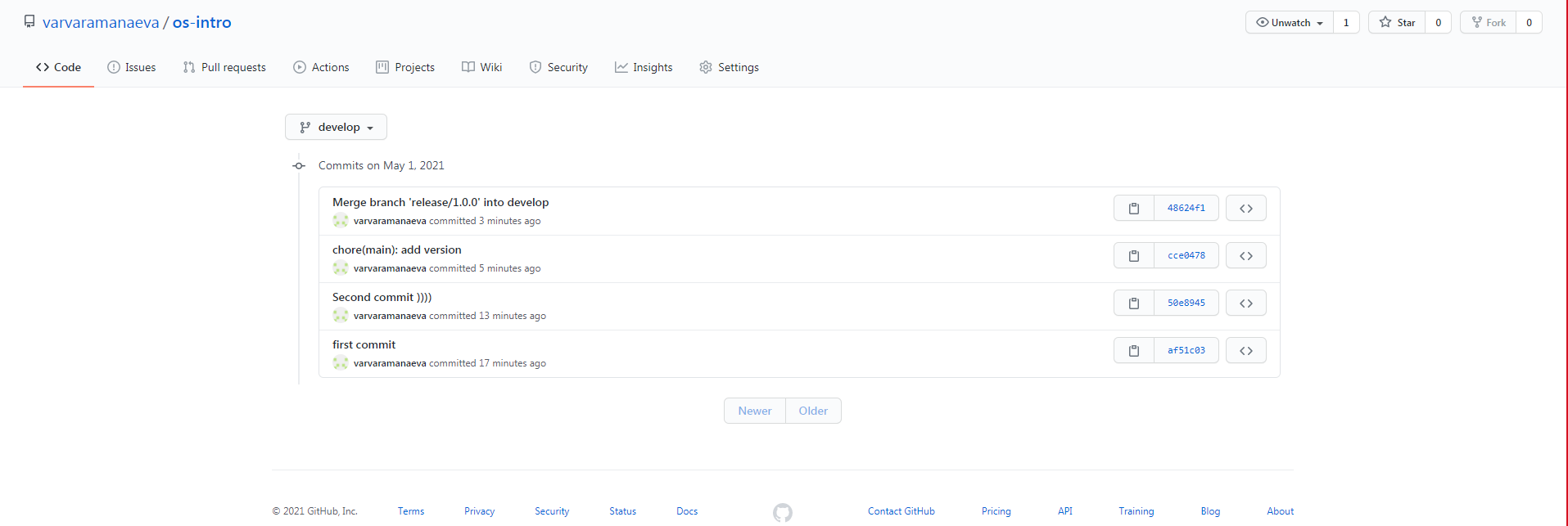
1. Был добавлен файл лицензии. Был добавлен шаблон игнорируемых файлов. Оба файла отправлены на *github.com*.

…



1. Инициализирован *git-flow*. Префикс версий записан в *v*. Записана версия в файл версии. Релизная ветка залита в основную ветку. Данные отправлены на *github.com*.

**Выводы:** были изучены идеология и применения средств контроля версий.

**Контрольные вопросы:**

*1) Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?*

Ответ: Система управления версиями (Version Control System, VCS) — программное обеспечение для облегчения работы с изменяющейся информацией. Системы контроля версий применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. Сохраняется также и история изменений проекта, дабы в любой момент изменения, повлекшие за собой ошибку и/или признанные ненужными, могли откатить до нужного этапа развития проекта.

*2) Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.*

Ответ: Хранилище – основное место хранения всех данных, к которому имеют доступ все члены проекта.  *Commit* – любое изменение к проекту/дополнение проекта.

История – последовательность *commit*-ов, то есть всех изменений/дополнений к проекту.

Рабочая копия – версия данных, которую пользователь выгружает для внесения изменений/доработки.

В хранилище хранятся история, все *commit*-ы с файлами и мельчайшими изменениями, а пользователь, желающий продолжить работу над проектом, вызывает себе рабочую копию (то есть, копирует информацию из тех *commit*-ов и версий, которые нужны ему / которые он планирует модифицировать).

*3) Что представляют собой и чем отличаются централизованные и децентрализованные* VCS*? Приведите примеры* VCS\* каждого вида.\*

Ответ: централизованные *VCS* – системы контроля версий, где в общем хранилище лежит репозиторий, из которого каждый пользователь берёт файлы, изменяет их и возвращает, так что меняются все файлы (откатиться до предыдущей версии в случае ошибки никто не запрещал). Таким видом *VCS* является *Subversion*.

Децентрализованные *VCS* – системы контроля версий, где у каждого пользователя может быть своя версия репозитория, и получить доступ можно ко всем версиям репозитория. Например, таким видом *VCS* является *Git*.

*4) Опишите действия с* VCS\* при единоличной работе с хранилищем.\*

Создаётся репозиторий. Туда отправляются первые файлы, образуя собой первую рабочую копию. Человек берёт рабочую копию, модифицирует её и выгружает уже обновлённую, создавая вторую версию проекта. Потом третью, четвёртую… Возможен откат к предыдущей версии в любой момент времени, так как даже при возврате к предыдущей версии достаточно сложно войти в противоречие с самим собой.

*5) Опишите порядок работы с общим хранилищем VCS.*

Создаётся репозиторий. Однако теперь необходимо добавить систему контроля версий, которая сможет разделять версии репозиториев разных людей, объединять их и устранять конфликты между версиями. Откаты к предыдущим версиям теперь должны согласовываться, различные изменения вносятся в различные файлы в разных ветвях хранилища дабы избежать конфликтов ещё на этапе изменения.

*6) Каковы основные задачи, решаемые инструментальным средством git?*

У *Git* две основных задачи: первая — хранить информацию о всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом.

*7) Назовите и дайте краткую характеристику командам git.*

*git init* – создание основного дерева репозитория.

*git pull* – получение обновлений (изменений) текущего дерева из центрального репозитория.

*git push* – отправка всех произведённых изменений локального дерева в центральный репозиторий.

*git status* – просмотр списка изменённых файлов в текущей директории.

*git diff* – просмотр текущих изменения.

Сохранение текущих изменений:

* *git add .* – добавить все изменённые и/или созданные файлы и/или каталоги.
* *git add имена\_файлов* – добавить конкретные изменённые и/или созданные файлы и/или каталоги.
* *git rm имена\_файлов* – удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории).

Сохранение добавленных изменений:

* *git commit -am ‘Описание коммита’* – сохранить все добавленные изменения и все изменённые файлы.
* *git commit* – сохранить добавленные изменения с внесением комментария через встроенный редактор.

*git checkout -b имя\_ветки* – создание новой ветки, базирующейся на текущей.

*git checkout имя\_ветки* – переключение на некоторую ветку (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой).

*git push origin имя\_ветки* – отправка изменений конкретной ветки в центральный репозиторий.

*git merge –no-ff имя\_ветки* – слияние ветки с текущим деревом.

Удаление ветки:

* *git branch -d имя\_ветки* – удаление локальной уже слитой с основным деревом ветки.
* *git branch -D имя\_ветки* – принудительное удаление локальной ветки.
* *git push origin :имя\_ветки* – удаление ветки с центрального репозитория.

*8) Приведите примеры использования при работе с локальным и удалённым репозиториями.*

В локальном репозитории мы используем *git pull* чтобы получить данные текущей версии, в локально же репозитории мы изменяем её состав (добавляем новый файл командой *git add*, например), а затем командой *git checkout имя\_ветки* переключаемся на нужную ветку в удалённом репозитории и командой *git push* отправляем новую версию на рассмотрение и проверку.

*9) Что такое и зачем могут быть нужны ветви (branches)?*

Ветки – различные каталоги в удалённом хранилище. Могут быть нужны в проектах со сложной структурой, где постоянно нужно контролировать изменения и где ошибки в последних версиях могут «уронить» всё приложение. В таких проектах ветви могут быть удобны для хранения различных вариантов изменений файлов (на всякий случай). Наиболее же удобными ветки являются в проектах, где много людей, которые работают над разными файлами системы. В таких проектах ветви – настоящее спасение от конфликтов между разными версиями разных файлов и в таких проектах разные ветви обычно принадлежат разным людям (реже группам) для наиболее удобной работы.

*10) Как и зачем можно игнорировать некоторые файлы при commit?*

Некоторые файлы при commit можно игнорировать, если использовать команду git add с конкретными (нужными) названиями файлов. Это может быть необходимо, если в проект нет необходимости загружать исполнительные файлы, или какие-то промежуточные файлы, которые не несут в себе никакой пользы для итоговой рабочей версии, но использовались при написании файла и могут быть востребованы при продолжении разработки.