**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

*дисциплина: Операционные системы*

Студент: Аристова Арина Олеговна

Студ. билет: 1032216433

Группа: НФИбд-01-21

**МОСКВА**

2022 г.

**Цель работы:**

* Изучить идеологию и применение средств контроля версий.
* Освоить умения по работе с git.

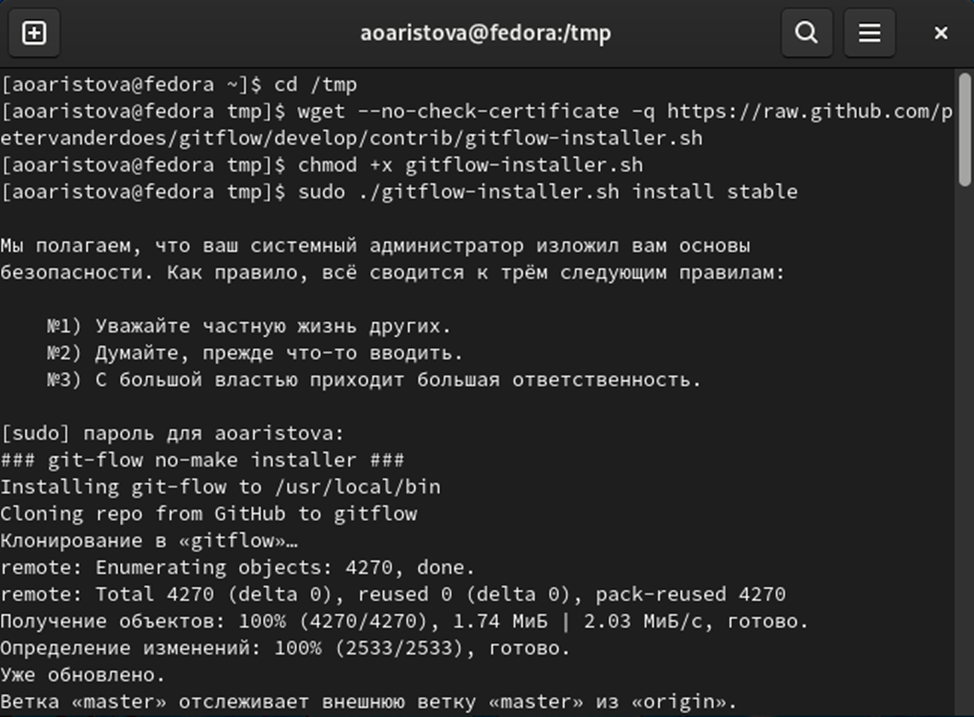
**Задание:**

* Создать базовую конфигурацию для работы с git;
* Создать ключ SSH;
* Создать ключ PGP;
* Настроить подписи git;
* Зарегистрироваться на Github;
* Создать локальный каталог для выполнения заданий по предмету.

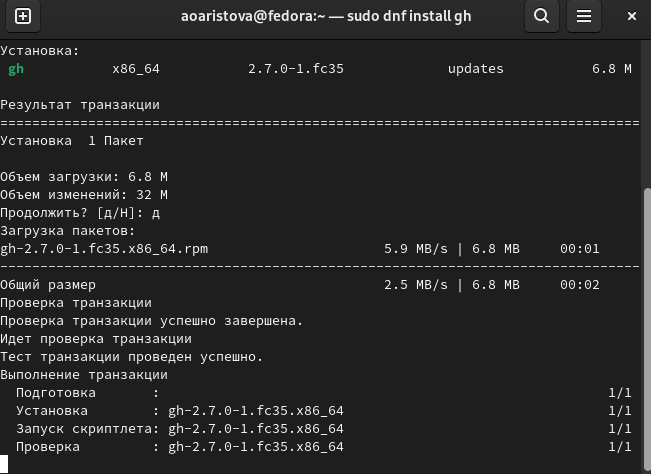
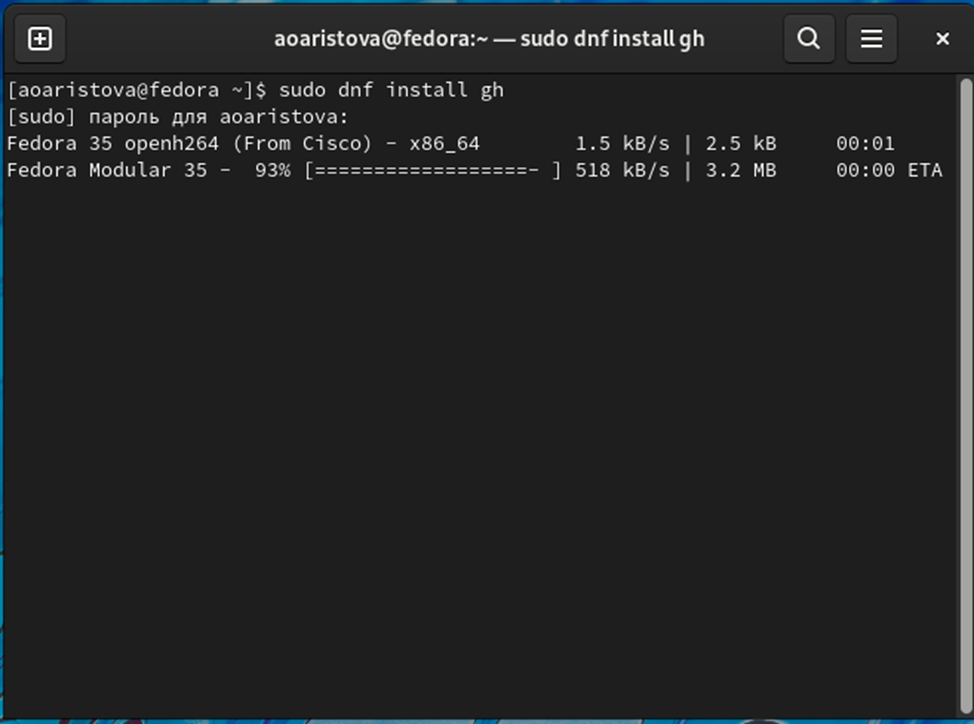
**Выполнение работы:**

Предварительно создаю учетную запись и заполняю основные данные на https://github.com.

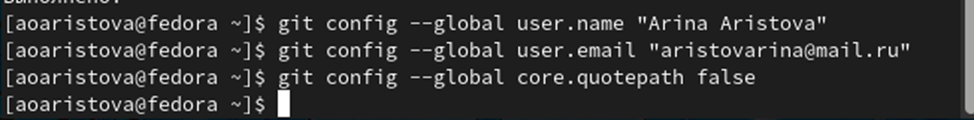
Устанавливаю git-flow в Fedora Linux.



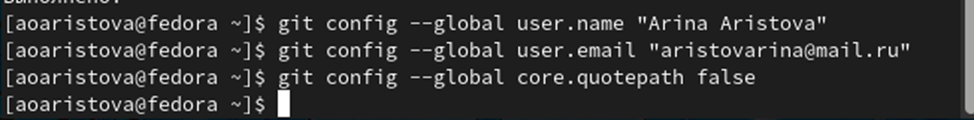
Устанавливаю gh в Fedora Linux.



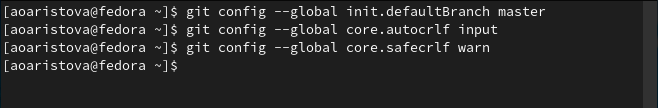
Выполняю базовую настройку git.



Настраиваю utf-8 в выводе сообщений git:

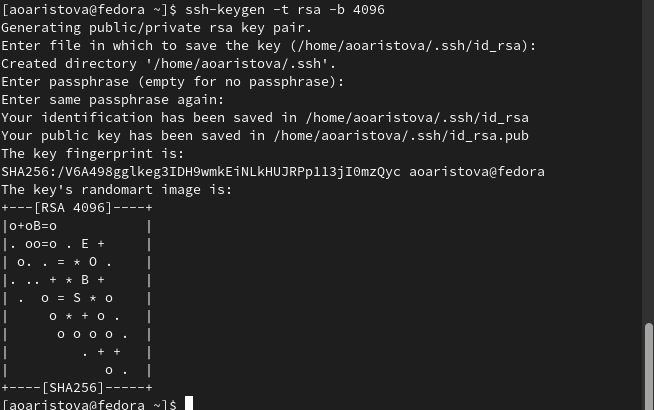


Настраиваю верификацию и подписание коммитов git. Задаю имя начальной ветки master, параметр autocrlf, параметр safecrlf.

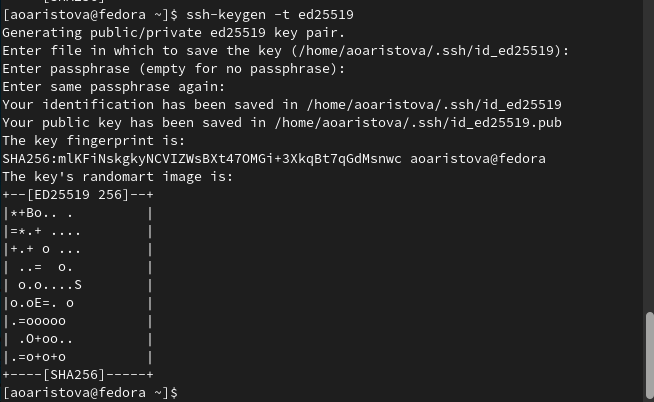


Создаю ключи ssh:

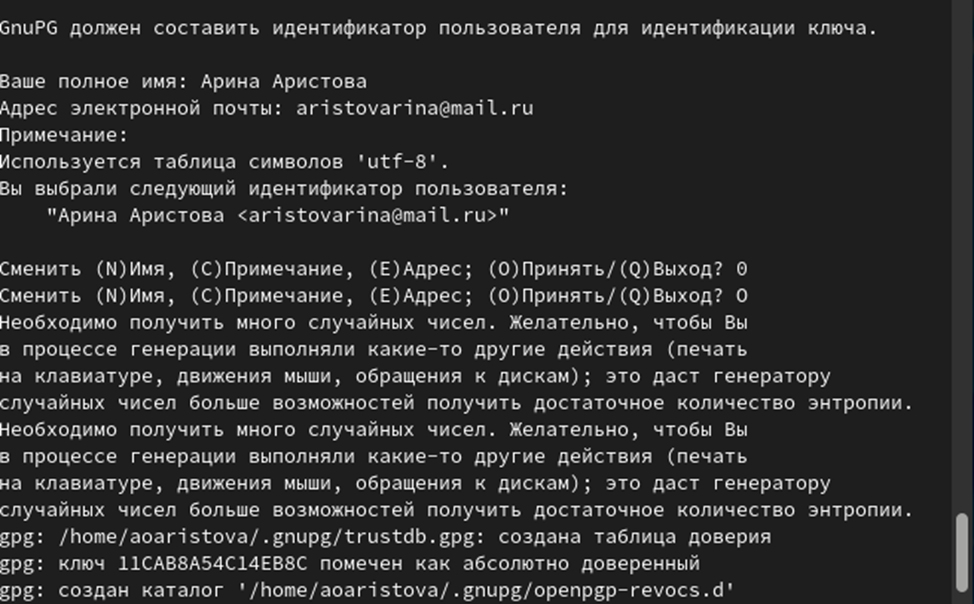
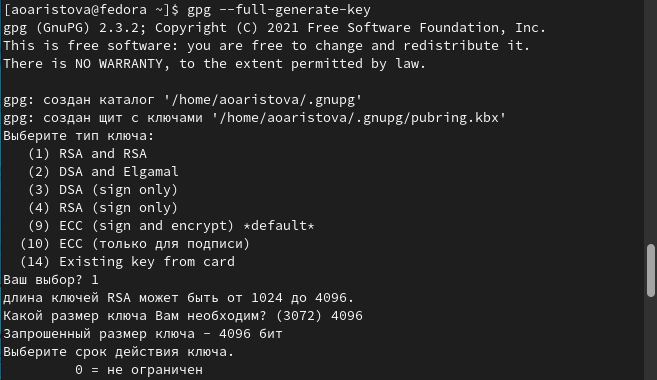
по алгоритму rsa с ключём размером 4096 бит;



по алгоритму ed25519.

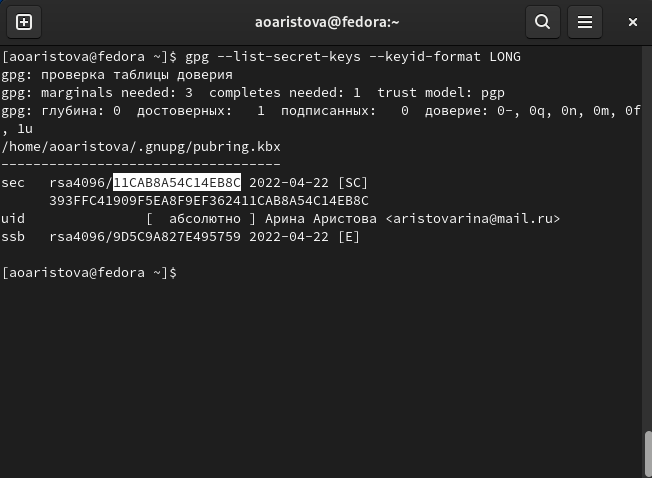


Создаю ключ pgp. Генерирую ключ. Из предложенных опций выбираю: тип RSA and RSA; размер 4096; срок действия - 0 (срок действия не истекает никогда), примечание оставляю пустым.

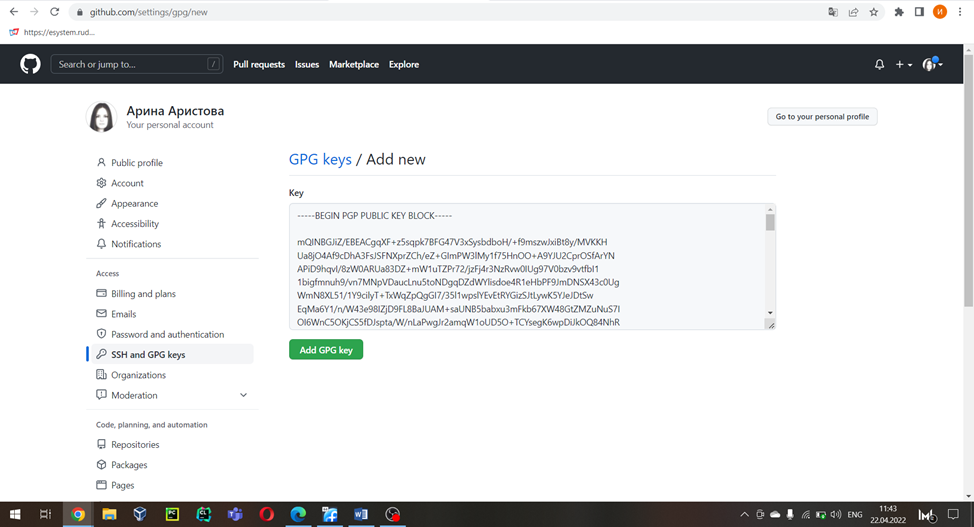


Добавляю PGP ключ в GitHub:

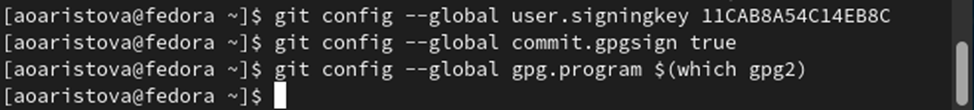
Вывожу список ключей и копирую отпечаток приватного ключа.



Копирую сгенерированный PGP ключ в буфер обмена и вставляю полученный ключ в поле ввода в GitHub.

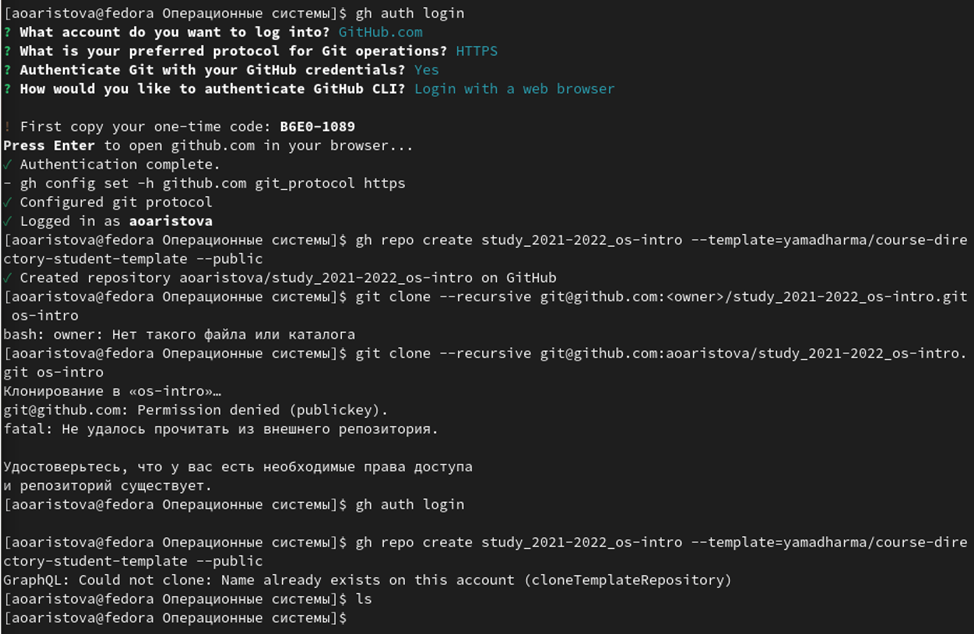


Настраиваю автоматические подписи коммитов git. Используя введённый email, указываю Git применять его при подписи коммитов.

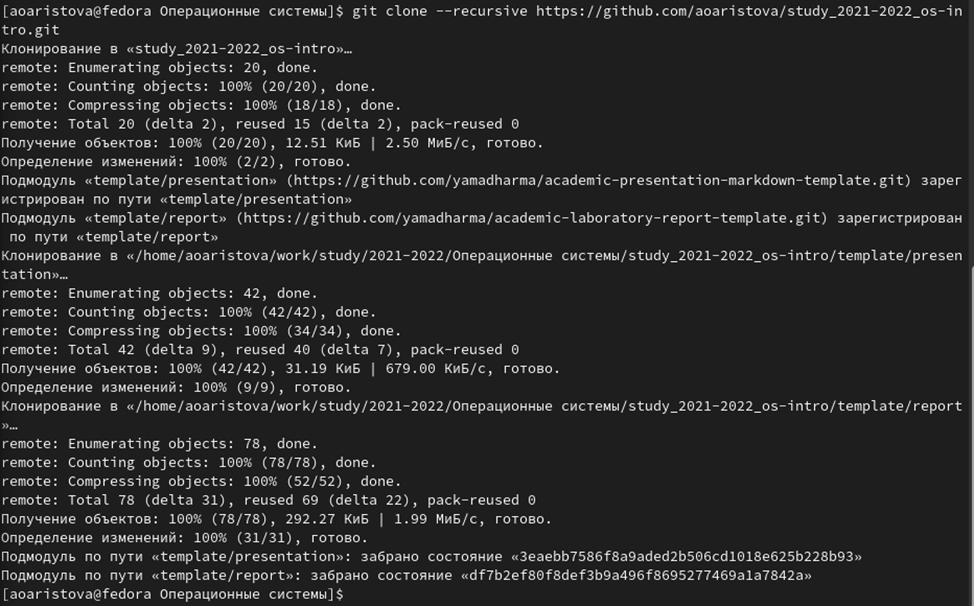
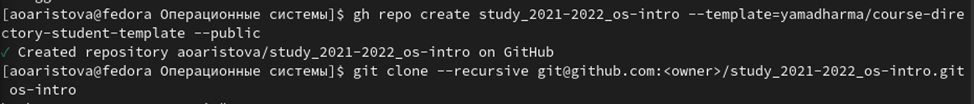


Настраиваю gh:

Авторизуюсь через браузер.

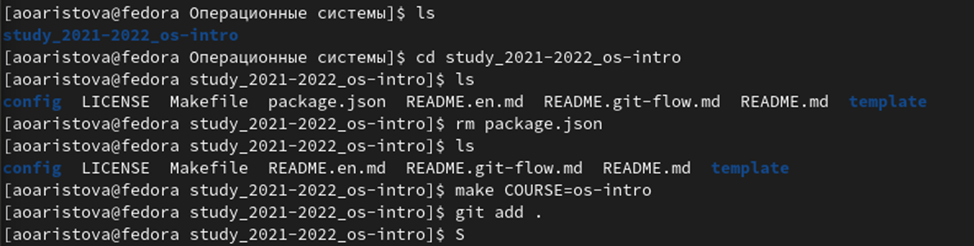


Создаю репозиторий курса на основе шаблона.



Настраиваю каталога курса.

Перехожу в каталог курса, удаляю лишние файлы, создаю необходимые каталоги и отправляю файлы на сервер.



**Вывод:** я изучила идеологию применения средств контроля версий, а также освоила умения по работе с git.

**Ответы на контрольные вопросы к лабораторной работе №2:**

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

VCS – (Version Control System) системы контроля версий. Они 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. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется.

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

Хранилище – место расположения файлов и папок проекта, изменения в которых отслеживаются.  
Commit – операция, предполагающая отправку в репозиторий изменений, которые пользователь внес в свою рабочую копию.  
Рабочая копия - текущее состояние файлов проекта, полученных из хранилища и, возможно, измененных, то есть разработчик имеет в распоряжении именно рабочую копию и с ней работает.

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

Централизованные VCS имеют одно основное хранилище. Каждый пользователь копирует себе необходимые файлы из основного репозитория, изменяет их, а затем добавляет изменения обратно (Subversion).  
Децентрализованные VCS устроены так, что каждый пользователь имеет свой (или даже не один) репозиторий. Пользователь может добавлять и забирать изменения из любого репозитория (Git).  
Соответственно, количеством основных репозиториев и различаются централизованные и децентрализованные VCS.

4. Опишите действия с VCS при единоличной работе с хранилищем.

Разработчик работает с веткой master, при необходимости может создать ветки для отдельных частей проекта. При завершении изменений разработчик *коммитит (commit)* и *пушит (push)* их, то есть сохраняет изменения в общем хранилище.

5. Опишите порядок работы с общим хранилищем VCS.

Каждый разработчик проекта работает над отдельной часть проекта в своей ветке. После завершения изменений разработчик *коммитит (commit)* и *пушит (push)* изменения на сервер. После окончания работы необходимо *смерджить(merge*), то есть выполнить слияние, веток, например, с главной веткой. Также разработчик может работать с изменениями, сделанными другим разработчиком, если на одной ветке их работает несколько.

6. Каковы основные задачи, решаемые инструментальным средством git?

Основные задачи git заключаются в удобной командной работе над проектом, а также в хранении информации обо всех изменениях проекта.

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 изменений;  
*git add* - сохранение текущих изменений: добавить все изменённые и/или созданные файлы и/или каталоги;  
*git add имена\_файлов* - сохранение текущих изменений: 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);  
*git commit -am 'Описание коммита'* - сохранение добавленных изменений:сохранить все добавленные изменения и все изменённые файлы;  
*git commit* - сохранение добавленных изменений: 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;  
*git push origin имя\_ветки* - отправка изменений конкретной ветки в центральный репозиторий;  
*git merge --no-ff имя\_ветки* - слияние ветки с текущим деревом;  
*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.

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

Ветка – указатель на один из коммитов. Ветки используются для разработки одной части проекта отдельно от других его частей. Каждая ветка представляет собой отдельную копию кода проекта. Ветки позволяют работать одновременно над разными версиями проекта.

10. Как и зачем можно игнорировать некоторые файлы при commit?

Игнорировать файлы при *commit* можно с помощью файла .gitignor . Туда обычно помещаются файлы, которые не нужны для проекта, например, временные файлы, создаваемые редакторами.