Front matter

lang: ru-RU title: "Отчет по лабораторной работе №2" subtitle: "по дисциплине: Операционные системы" author: "Трефилова Мария Андреевна"

Formatting

toc-title: "Содержание" toc: true # Table of contents toc_depth: 2 lof: false # List of figures lot: false # List of tables fontsize: 12pt linestretch: 1.5 papersize: a4paper documentclass: scrreprt polyglossia-lang: russian polyglossia-otherlangs: english mainfont: PT Serif romanfont: PT Serif sansfont: PT Sans monofont: PT Mono mainfontoptions: Ligatures=TeX romanfontoptions: Ligatures=TeX sansfontoptions: Ligatures=TeX,Scale=MatchLowercase monofontoptions: Scale=MatchLowercase indent: true pdf-engine: lualatex header-includes: - \linepenalty=10 # the penalty added to the badness of each line within a paragraph (no associated penalty node) Increasing the value makes tex try to have fewer lines in the paragraph. - \interlinepenalty=0 # value of the penalty (node) added after each line of a paragraph. - \hyphenpenalty=50 # the penalty for line breaking at an automatically inserted hyphen - \exhyphenpenalty=50 # the penalty for line breaking at an explicit hyphen - \binoppenalty=700 # the penalty for breaking a line at a binary operator - \relpenalty=500 # the penalty for breaking a line at a relation - \clubpenalty=150 # extra penalty for breaking after first line of a paragraph - \widowpenalty=150 # extra penalty for breaking before last line of a paragraph - \displaywidowpenalty=50 # extra penalty for breaking before last line before a display math - \brokenpenalty=100 # extra penalty for page breaking after a hyphenated line - \predisplaypenalty=10000 # penalty for breaking before a display - \postdisplaypenalty=0 # penalty for breaking after a display - \floatingpenalty = 20000 # penalty for splitting an insertion (can only be split footnote in standard LaTeX) - \raggedbottom # or \flushbottom - \usepackage{float} # keep figures where there are in the text

- \floatplacement{figure}{H} # keep figures where there are in the text

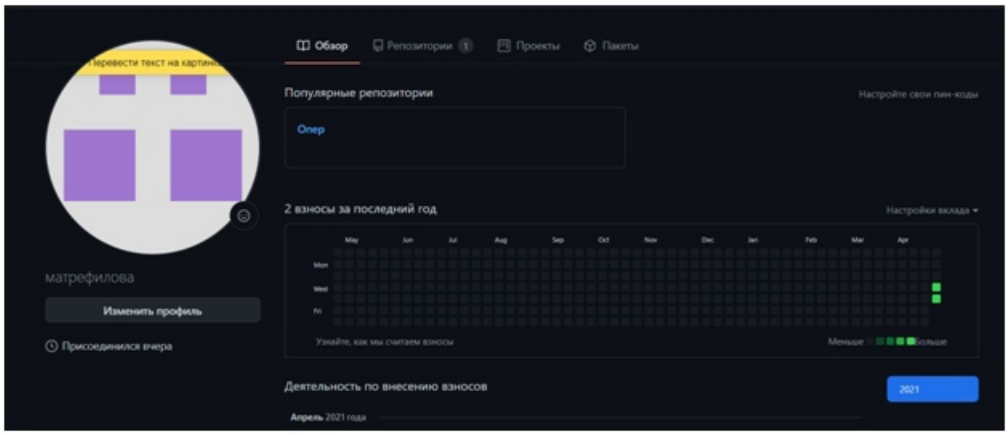
Цель работы

Изучить идеологию и применение средств контроля версий.

Выполнение лабораторной работы

Настройка git

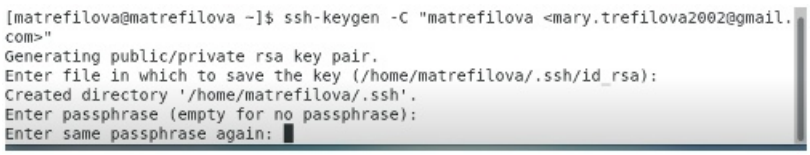
1. Создаём учётную запись на https://github.com.

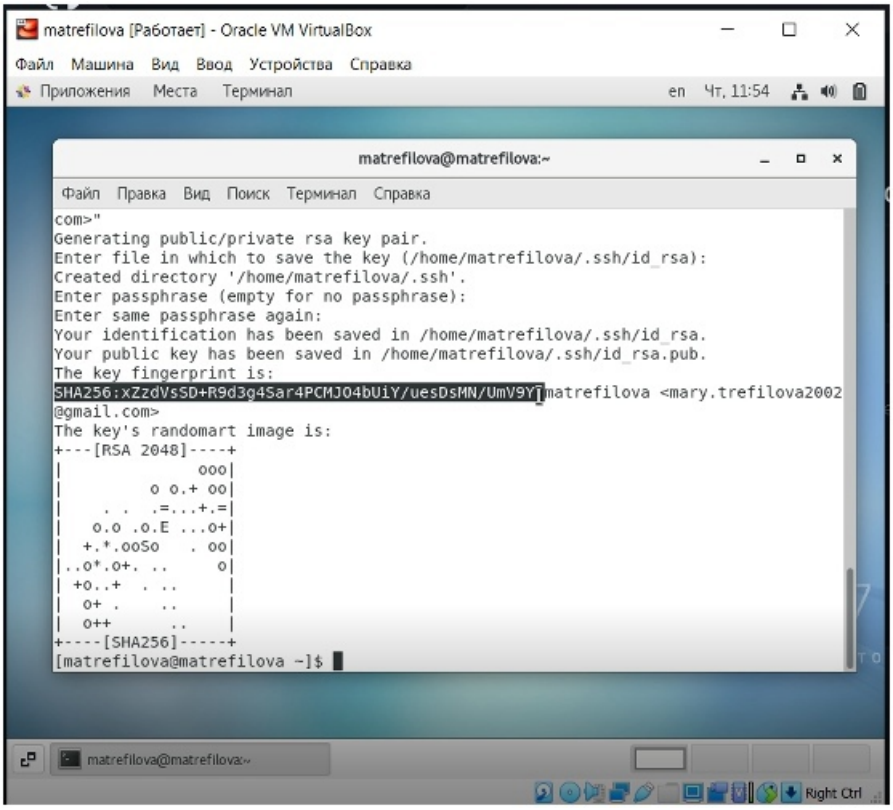


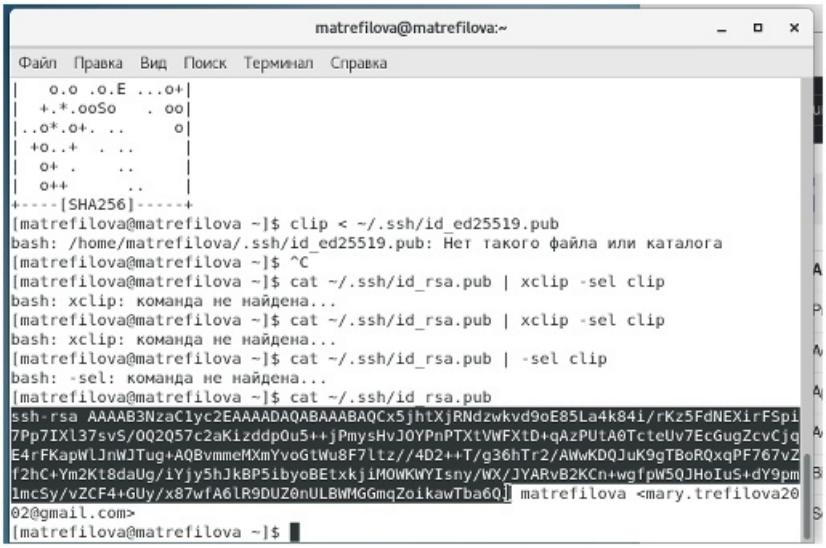
2. Настраиваю систему контроля версий git, как это описано в памятке к лабораторной работе, с использованием сервера репозиториев https://github.com/.

    1. Генерируем пару ключей (приватный и открытый):

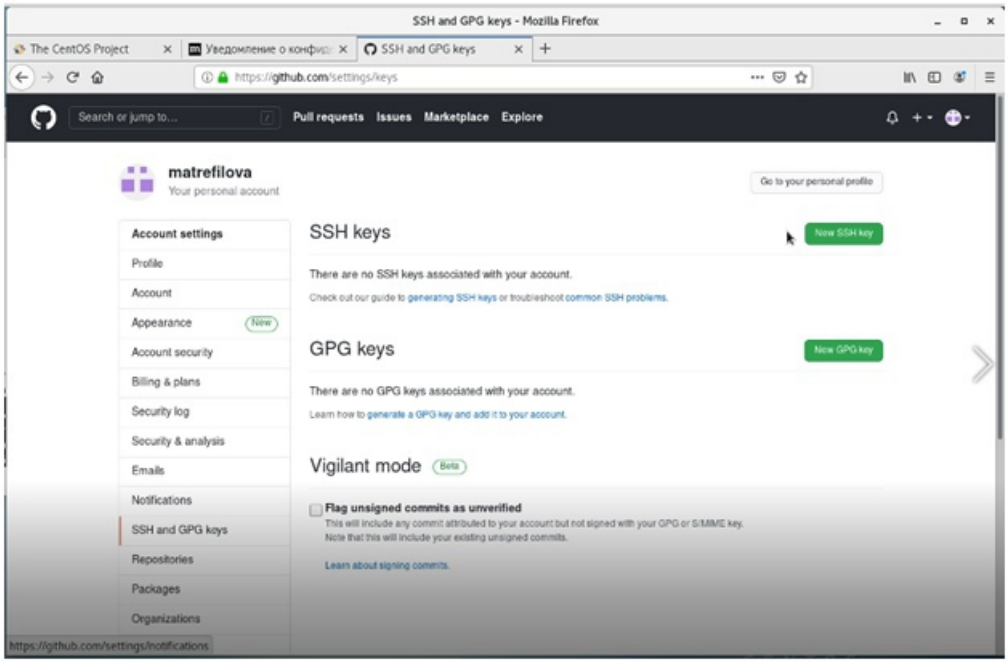
```
ssh-keygen -C "matrefilova 1032201663@pfur.ru"
```







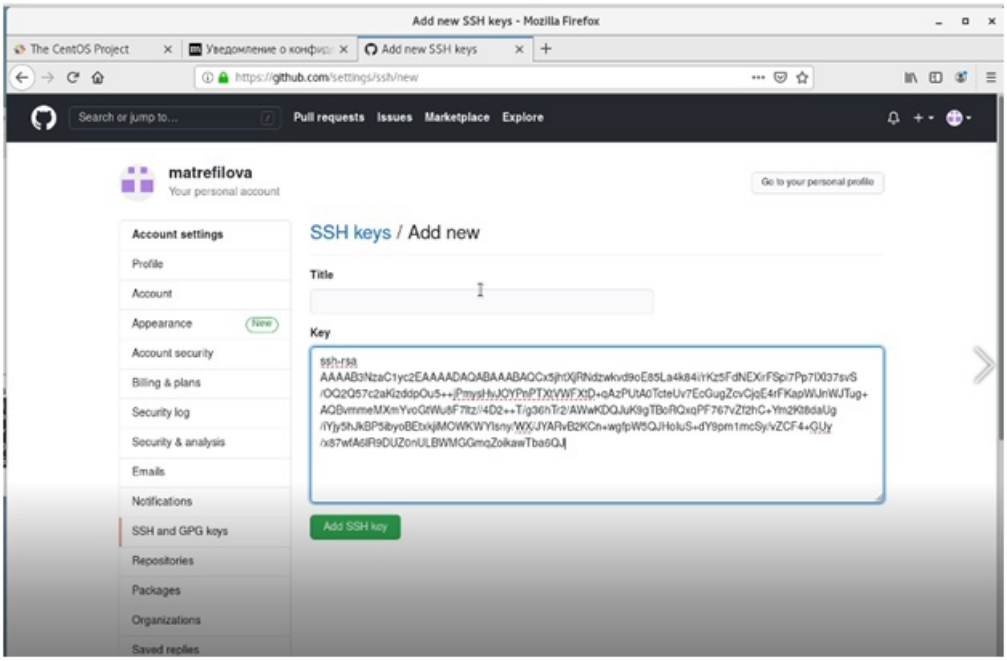
2. Загружаем сгенерированный нами ранее открытый ключ. Заходим на сайт https://github.com/ под своей учётной записью и переходим в меню GitHub setting. После этого выбираем в боковом меню GitHub setting SSH-ключи. Нажимаем кнопку Добавить ключ.



Копируем из локальной консоли ключ в буфер обмена с помощью команды:

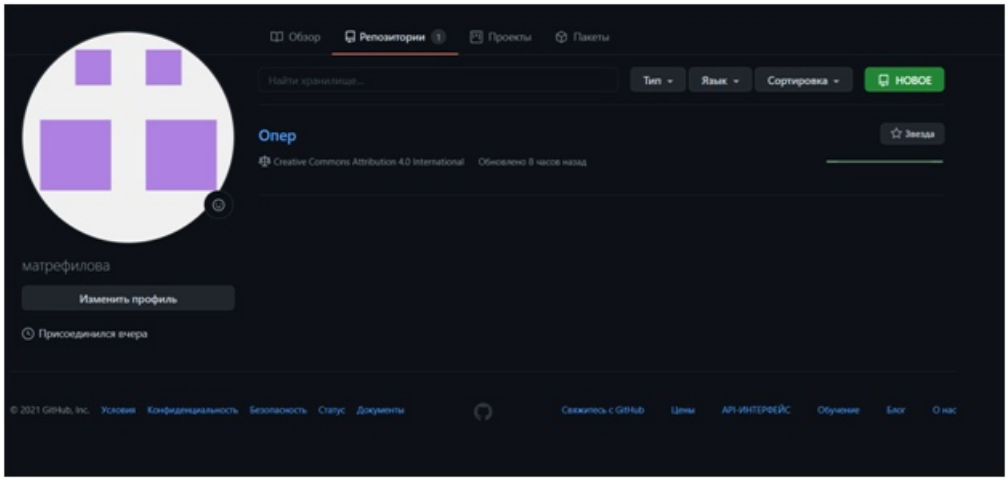
```
cat ~/.ssh/id_rsa.pub | xclip -sel clip
```

Вставляем ключ в появившееся на сайте поле.

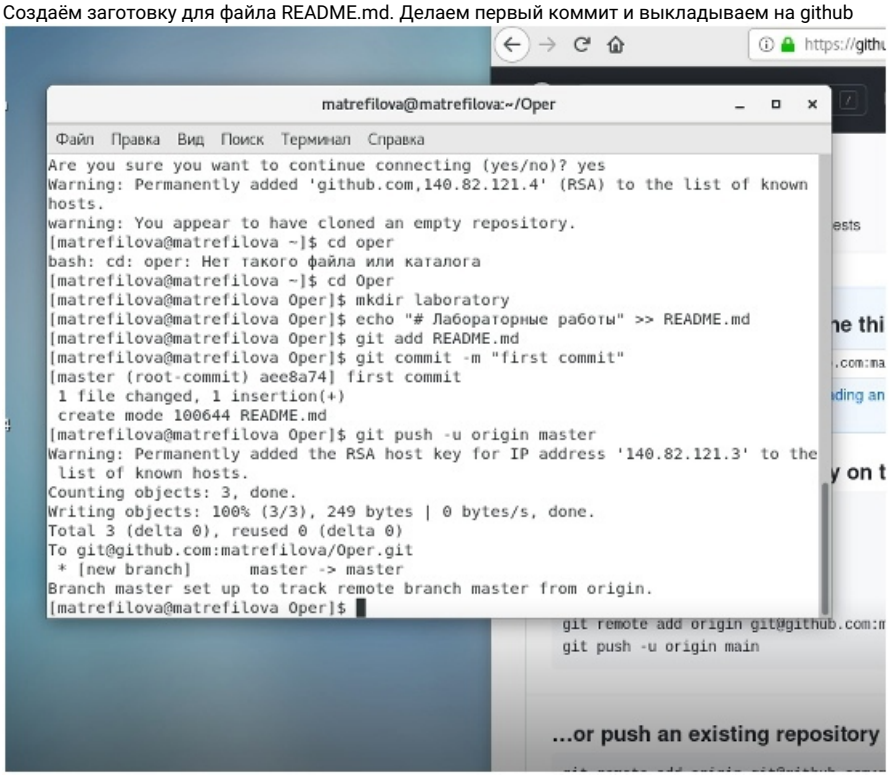


Подключение репозитория к github

1. Создаём репозиторий на GitHub. Назовём его Oper



2. Создаём заготовку для файла README.md. Делаем первый коммит и выкладываем на github



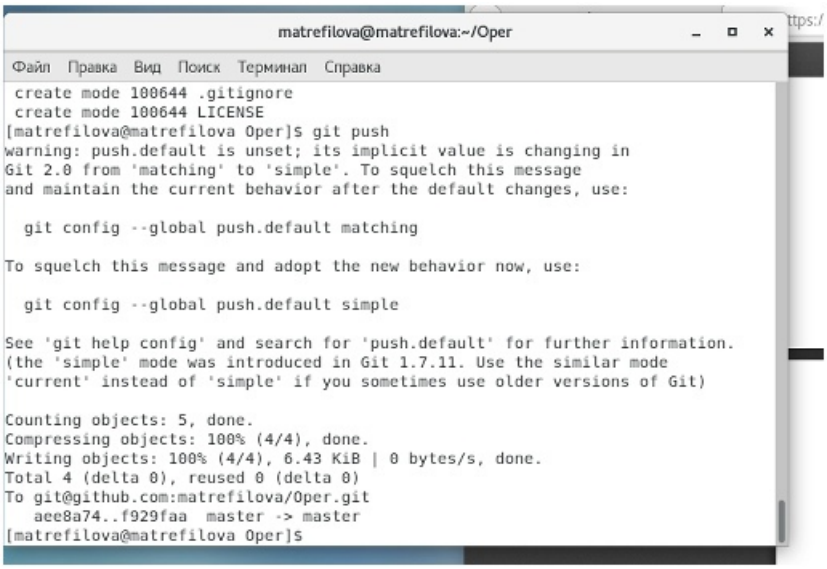
3. Добавим файл лицензии

```
[matrefilova@matrefilova Oper]$ wget https://creativecommons.org/licenses/by/4.0
/legalcode.txt -O LICENSE
--2021-04-29 13:06:01--  https://creativecommons.org/licenses/by/4.0/legalcode.t
xt
Распознаётся creativecommons.org (creativecommons.org)... 104.20.150.16, 172.67.
34.140, 104.20.151.16, ...
Подключение к creativecommons.org (creativecommons.org)|104.20.150.16|:443... со
единение установлено.
HTTP-запрос отправлен. Ожидание ответа... 200 OK
Длина: нет данных [text/plain]
Сохранение в: «LICENSE»

    [ <=>                                   ] 18 657      --.-K/s   за 0,01s

2021-04-29 13:06:03 (1,71 MB/s) - «LICENSE» сохранён [18657]

[matrefilova@matrefilova Oper]$
```

2. Добавим шаблон игнорируемых файлов. Просмотрим список имеющихся шаблонов:

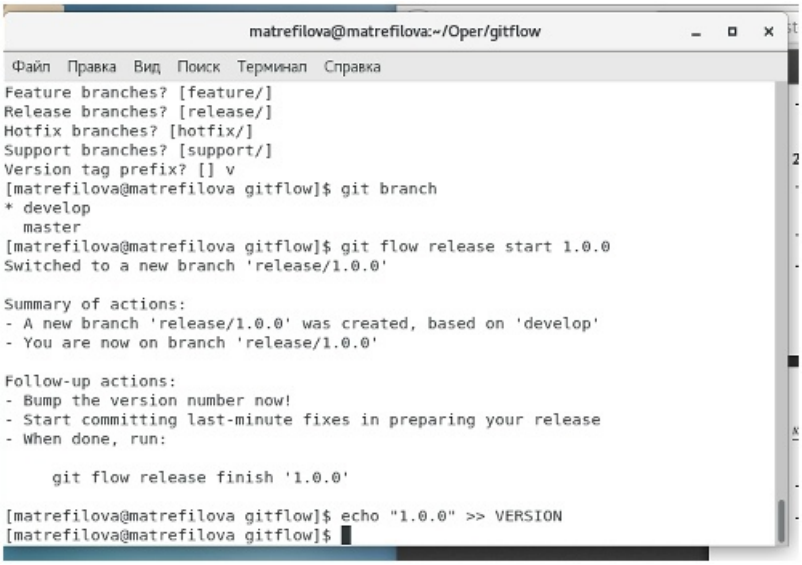
```
[matrefilova@matrefilova Oper]$ curl -L -s https://www.gitignore.io/api/list
```

3.
   - Добавляю новые файлы: git add .
- Выполняю коммит: git commit -a
- Отправляю на github: git push

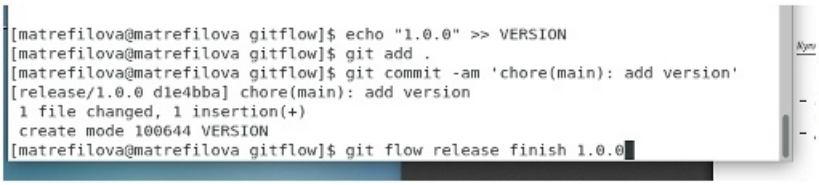


Конфигурация git-flow

1. Инициализируем git-flow. Проверьте, что Вы на ветке develop: git branch – Создадим релиз с версией 1.0.0. Запишем версию.



2. Добавим в индекс. Зальём релизную ветку в основную ветку. Отправим данные на github.



Выводы

Таким образом, я изучила идеологию и применение средств контроля версий

Контрольные вопросы

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

Система контроля версий (VCS) — это место хранения кода. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы.

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

- Репозиторий - хранилище версий - в нем хранятся все документы вместе с историей их изменения и другой служебной информацией
- Commit («[трудовой] вклад», не переводится) — процесс создания новой версии
- Рабочая копия (working copy) — текущее состояние файлов проекта, основанное на версии, загруженной из хранилища (обычно на последней).
- Версия (revision), или ревизия, — состояние всех файлов на определенный момент времени, сохраненное в репозитарии, с дополнительной информацией

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

- Централизованные системы — это системы, которые используют архитектуру клиент / сервер, где один или несколько клиентских узлов напрямую подключены к центральному серверу. (Пример — Wikipedia.) Централизованная модель предполагает наличие единого репозитория для хранения файлов.
- В децентрализованных системах каждый узел принимает свое собственное решение. Конечное поведение системы является совокупностью решений отдельных узлов.(Пример — Git и Mercurial) Децентрализованные модели позволяют хранить репозиторий (его копию) у каждого разработчика, работающего с данной системой. При этом можно выделить центральный репозиторий (условно), в который будут отправляться изменения из локальных и, с ним же эти локальные репозитории будут синхронизироваться.

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

Например, можно делать commit, хранить файлы проекта в репозитории, делать релизы.

5. Опишите порядок работы с общим хранилищем VCS.

Участник проекта (пользователь) перед началом работы посредством определённыхкоманд получает нужную ему версию файлов.После внесения изменений,пользователь размещает новую версию в хранилище. Можно объединить (слить) изменения,сделанные разными участниками (автоматически или вручную),вручную выбрать нужную версию,отменить изменения вовсеили заблокировать файлы для изменения.

6. Каковы основные задачи,решаемые инструментальным средством git?

У Git есть две основные задачи: хранить информацию обо всех изменениях в коде, начиная с самой первой строчки, и обеспечить удобства командной работы над кодом.

7. Назовите и дайте краткую характеристику командам git.

```
 1. создание основного дерева репозитория:
```
```
git init
```
```
2. получение обновлений (изменений) текущего дерева из центрального репозитория:
```
```
git pull
```
```
3. отправка всех произведённых изменений локального дерева в центральный репозиторий:
```
```
git push
```
```
4. просмотр списка изменённых файлов втекущей директории:
```
```
git status
```
```
5. просмотртекущих изменения:
```
```
git diff
```
```
6. добавить все изменённые и/или созданные файлы и/или каталоги:
```
```
git add .
```
```
7. добавить конкретные изменённые и/или созданные файлы и/или каталоги:
```
```
git add имена_файлов
```
```
8. удалить файли/или каталог из индекса репозитория(при этом файл/или каталог остаётся в локальной директории):
```
```
git rm имена_файлов
```
```
9. сохранить все добавленные изменения и все изменённые файлы:
```
```
git commit -am 'Описание коммита'
```

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

1. Если в проекте закрыт исходный код, то здесь удобно использовать локальный репозиторий. Также локальный репозиторий удобен для одиночного проекта.

2. Проекты с открытым кодом или выполняемые группой людей удобно хранить на удаленном репозитории.

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

Git branch’ – это команда для управления ветками в репозитории Git. Ветка – это просто «скользящий» указатель на один из коммитов. Когда мы создаём новые коммиты, указатель ветки автоматически сдвигается вперёд, к вновь созданному коммиту. Ветки используются для разработки одной части функционала изолированно от других. Каждая ветка представляет собой отдельную копию кода проекта. Ветки позволяют одновременно работать над разными версиями проекта.

Ветвление («ветка», branch) — один из параллельных участков истории в одном хранилище, исходящих из одной версии (точки ветвления). Ветки нужны для того, чтобы программисты могли вести совместную работу над проектом и не мешать друг другу при этом.

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

Игнорируемые файлы обычно представляют собой файлы, специфичные для платформы, или автоматически созданные из сборочных систем. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Временно игнорировать изменения в файле можно командой: git update-index --assume-unchanged