Семинар #1: Основы git. Практика.

Задача 1. Hello

Напишите `bash`-скрипт, который должен делать следующее:

1. Писать на экране сообщение `Hello World!`.
2. Создавать файл `hello.txt`, в котором будет записано `Hello World!`.

Протестируйте ваш `bash`-скрипт, запустив его.

Задача 2. Создайте папку с файлами

Напишите `bash`-скрипт, который должен делать следующее:

1. Создавать папку `animals` в текущей директории.
2. В этой папке создавать 2 файла: `cat.txt` и `dog.txt`.
3. В файл `cat.txt` нужно записать строку `"I am Cat!"`, а в файл `dog.txt` записать строку `"I am Dog!"`.
4. После исполнения скрипта в `bash` пользователь должен остаться в той же директории, в которой скрипт был запущен.
5. Писать сообщение об успешном завершении.

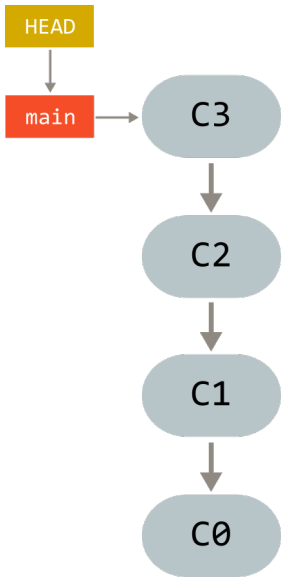
Протестируйте ваш `bash`-скрипт, запустив его.

Задача 3. Репозиторий из четырёх коммитов

Напишите `bash`-скрипт, который должен делать следующее:

1. Инициализировать новый пустой git репозиторий.
2. Создавать новый файл `cat.txt` и добавлять коммит в репозиторий, содержащий этот файл.
3. Создавать новый файл `dog.txt` и добавлять коммит, содержащий файл `dog.txt`.
4. Изменять файл `cat.txt` и добавлять коммит, содержащий эти изменения.
5. Удалять файл `dog.txt` и добавлять коммит, который уже не содержит файл `dog.txt`.
6. Сообщения коммитов должны корректно описывать происходящее.
7. Печатать информацию о всех сделанных коммитах в следующем формате:

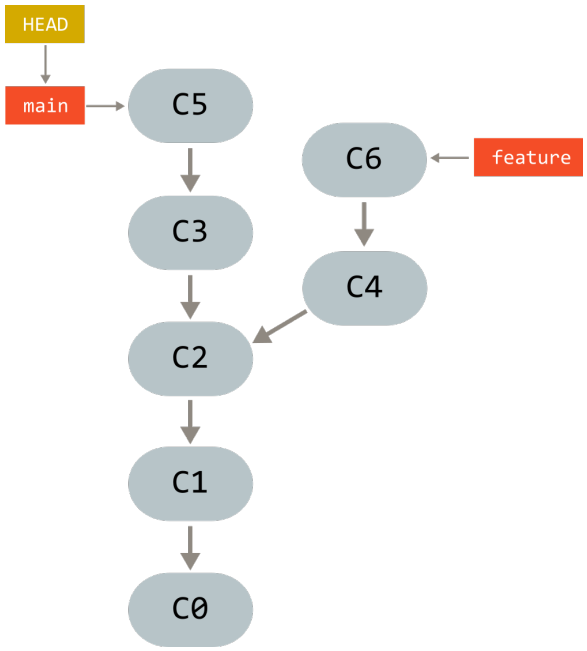
```
$ git log --oneline --all --graph
```



Протестируйте ваш `bash`-скрипт, запустив его.

Задача 4. Две ветки

Создайте новый локальный репозиторий и добавьте в него коммиты, таким образом, чтобы граф коммитов выглядел так, как это представлено на рисунке:



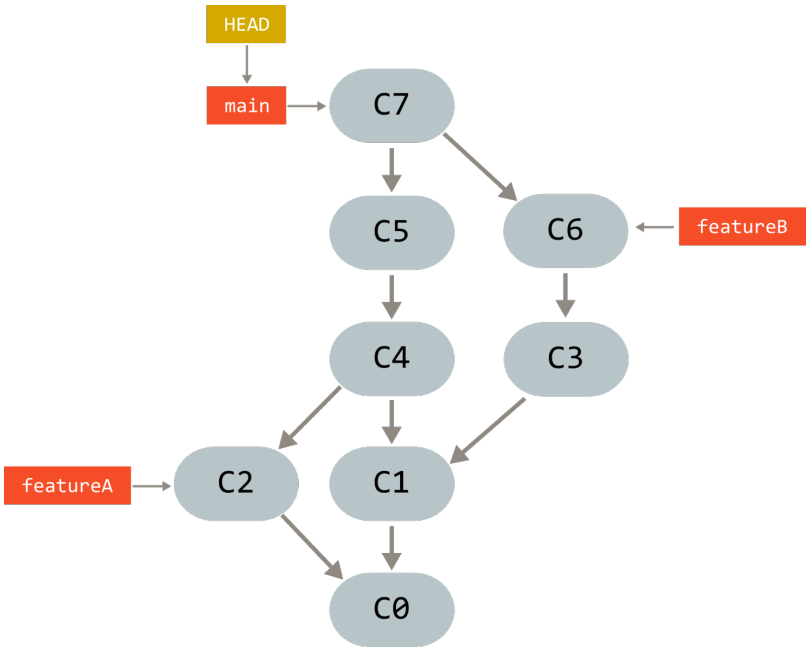
В репозитории должно быть две ветки: `main` и `feature`. Порядок добавления коммитов должен соответствовать порядковым номерам, изображенным на рисунке. Сообщения коммитов должны начинаться на их обозначения, изображенным на рисунке (`C0`, `C1` и т. д.). Конкретное содержимое файлов репозитория может быть любым – на ваш выбор. Для просмотра графа коммитов используйте:

```
$ git log --oneline --all --graph
```

Создайте bash-скрипт, который бы содержал все команды из этого задания (он должен с нуля создавать репозиторий, представленный на изображении). Протестируйте этот bash-скрипт.

Задача 5. Граф со слияниями

Сделать всё то же самое, что и в предыдущей задаче, но граф коммитов должен выглядеть так:



После этого удалите ветки `featureA` и `featureB` отдельными командами.

Задача 6. Слияние с конфликтом

Для проверки числа на простоту на языке C была написана функция `is_prime`. Эта функция находится в файле `prime.c` в репозитории `mipt-hsse.gitlab.yandexcloud.net/v.biryukov/prime_calculation`. В какой-то момент файл `prime.c` выглядел следующим образом:

```c
#include <stdio.h>

int is_prime(int n)
{
    if (n < 2)
        return 0;

    for (int i = 2; i < n; ++i)
    {
        if (n % i == 0)
            return 0;
    }
    return 1;
}

int main()
{
    int a = 25263551;

    if (is_prime(a))
        printf("%i is prime\n", a);
    else
        printf("%i is NOT prime\n", a);
}
```

Над этим проектом работала команда программистов, и в определённый момент практически одновременно произошли следующие события:

- Программист Алиса заметила, что алгоритм проверки числа на простоту не оптимален и его можно ускорить, если производить итерации не до n, а до $\sqrt{n}$. Алиса создала новую ветку `alice` и добавила в него коммит, с такой оптимизацией.
- В это же самое время программист Боб заметил, что алгоритм можно ускорить, если в процессе итерирования пропускать чётные числа (однако Боб не заметил оптимизацию, которую заметила Алиса). Боб создал новую ветку `bob` и добавил в него коммит, со своей оптимизацией.
- Параллельно с этим другие участники команды продолжали развивать проект. В главной ветке `main` появился новый коммит, расширяющий набор тестов в функции `main`.

Ваша задача – объединить все изменения из веток `alice` и `bob` в ветку `main`. После успешного объединения необходимо удалить ветки `alice` и `bob`. Затем нужно создать репозиторий на GitLab и отправить туда ваш репозиторий.

Решите эту задачу двумя способами:

1. Используя слияние – `git merge`.
2. Используя перебазирование – `git rebase`.

То есть в итоге на GitLab нужно будет создать 2 репозитория `prime_calculation_merge` и `prime_calculation_rebase`. В одном будет результат выполнения задания с помощью `git merge`, а в другом – с помощью `git rebase`.

Граф репозитория можно посмотреть на GitLab, если на странице репозитория нажать на левой панели `Code -> Repository graph`.

Задача 7. HEAD

Напишите bash-скрипт, который бы создавал новый git-репозиторий и приводил бы его в состояние `detached HEAD`.

Задача 8. Просмотр репозитория библиотеки stb

STB – это набор header-only библиотек на C/C++, распространяемых в виде исходного кода. Библиотека предоставляет минималистичные решения для распространённых задач: загрузки и сохранения изображений в разных форматах (`.jpg`, `.png` и другие), работы со шрифтами, декодирования аудио и обработки данных. Её ключевые преимущества – простота интеграции, отсутствие внешних зависимостей и удобство использования. STB идеально подходит для быстрого прототипирования, небольших проектов и сценариев, где не требуются сложные специализированные библиотеки.

Репозиторий проекта можно найти на GitHub: `github.com/nothings/stb`. Вам нужно клонировать этот репозиторий себе и произвести следующие операции:

1. Просмотрите всю историю коммитов с помощью команды:

   ```
   $ git log --oneline --all --graph
   ```

   Удобнее смотреть в файле:

   ```
   $ git log --oneline --all --graph > history.txt
   ```

2. Выполните команду, которая бы печатала информацию о всех коммитах, сделанных с начала текущего года.

3. Выполните команду, которая бы печатала информацию о всех коммитах, в которых менялся файл `stb_image.h`.

4. Выполните команду, которая бы печатала информацию о всех коммитах, в которых изменялась функция `stbi_jpeg_load` из файла `stb_image.h`.

5. Используйте `git switch`, чтобы перейти на второй коммит в истории проекта. Создайте новую ветку, указывающую на этот коммит.

6. Используйте `git rebase` чтобы "выпрямить" историю коммитов.

7. Используйте интерактивный `git rebase` чтобы сжать всю историю до 10-ти коммитов. Выберете коммиты равномерно в истории проекта.

Задача 9. Задание совместную работу с репозиторием

Сделайте задание по адресу `mipt-hsse.gitlab.yandexcloud.net/pro100savant/hellow_word_2025`.