**Лабораторная работа**

**Работа с GIT**

<https://git-scm.com/book/ru/v2>

https://proglib.io/p/git-cheatsheet

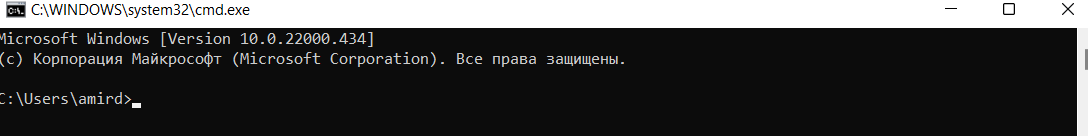
<https://proglib.io/p/10-tips-git>

**Задания:.**

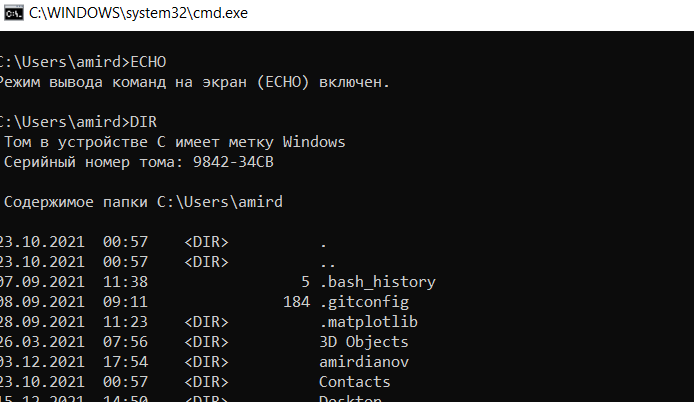
Устанавливаем git, если нет . Работаем только через командную строку. Все результаты работы фиксируете в отчете в виде скриншотов после каждого пункта.

А) Основа

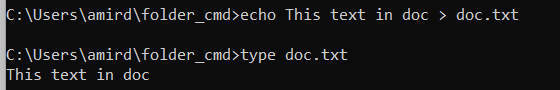
1. Работа в командной строке

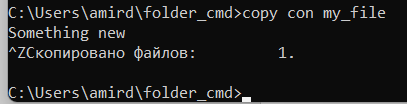


1. Посмотреть справку по команде(не менее двух способов).



1. Создать пустой файл с указанным именем (не менее двух способов).





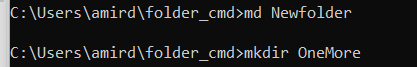
После copy con открывается поле редактирования и чтобы сохранить ctrl +z и затем выти ctrl + c

1. Создать файл с заданным атрибутом.





1. Создать папку.



1. Скопировать указанный файл в другую папку.



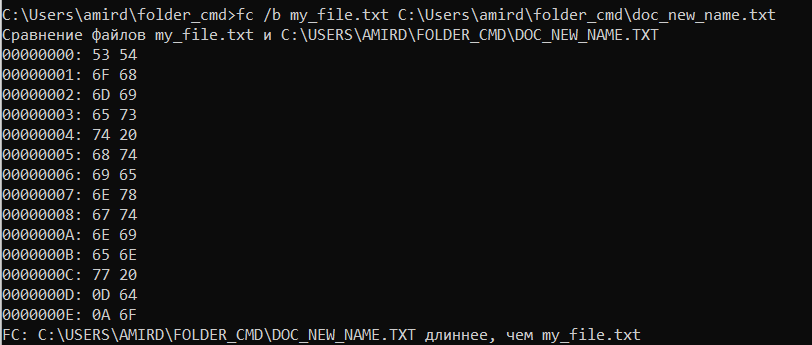
1. Скопировать файлы с указанным расширением в другую папку.
2. Переименовать файл.



1. Переместить файл.



1. Сравнить два файла (набора файлов) и вывести различия между ними.

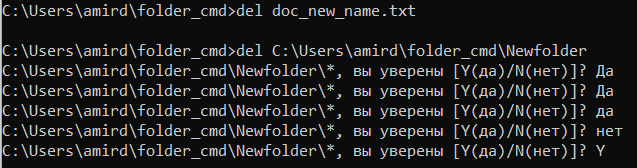


1. Удалить файл.



1. Удалить папку.

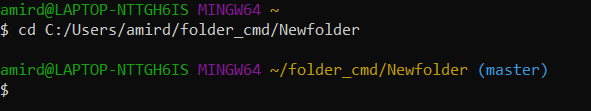


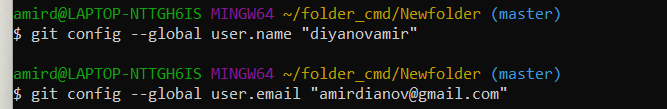


1. Создать локальный git-репозиторий с именем hello-world



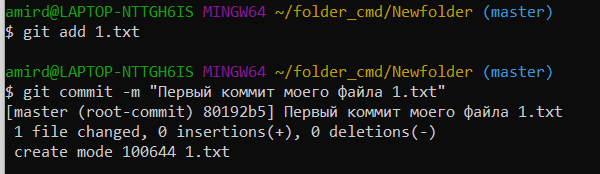
1. Провести инициализацию репозитория в созданной папке. Для этого, открыть программу Git Bash, перейти в созданную папку. Установить настройки имени и e-mail’а.



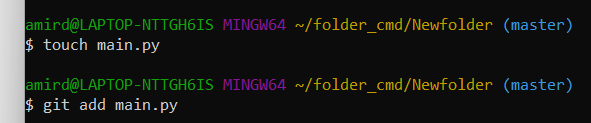


1. Создать текстовый файл 1.txt. Сформируйте этот документ, создавая commit для каждого абзаца.

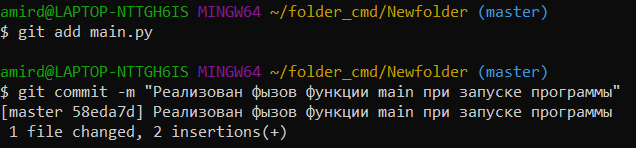




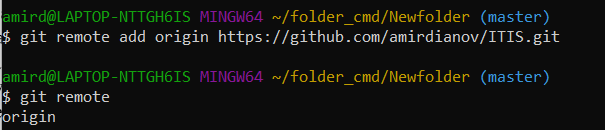
1. Разработать приложение Hello, World. Работу выполнить в несколько коммитов. Например:
2. Создать пустой файл main.py
3. Добавить функцию main



1. Реализовать вывод строки «Hello, World»

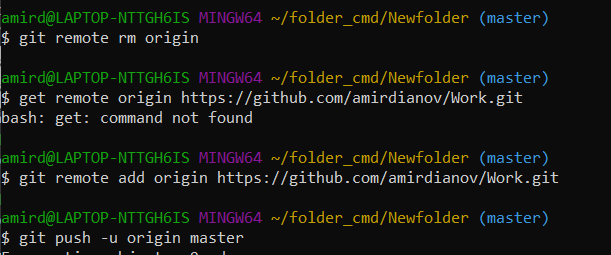


1. Создать проект на GitHub
2. Создать удаленный репозиторий (на github). Cвязать локальный репозиторий с проектом на GitHub.



1 – связывание

2 – проверка

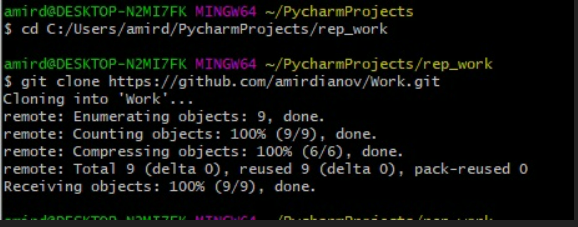


1 – удаление remote

2 – связывание

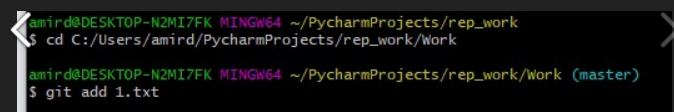
3 – запушить

1. Добавить к удаленному репозиторию участников проекта (соседей по парте).   
     
   но изначально нужно перейти в своем репозитории в гитхабе   
   settings  
   colloborators  
   и добавить товариша  
   после того только он сможет запушить в твой гит с его локального репа

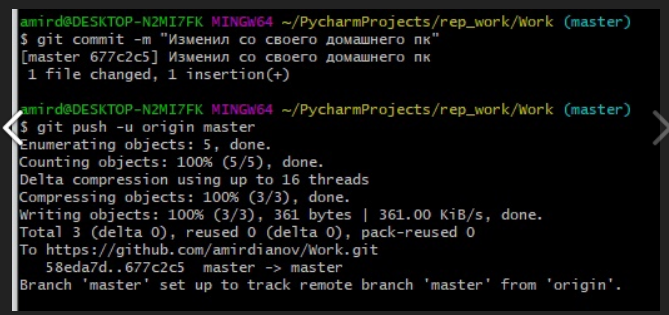


Мой товарищ клонирует себе удаленный репозиторий, тем самым создавая у себя локальный репозиторий

1. Получить данные из удаленного репозитория (выполняется прочими студентами).
2. Изменить полученные данные. Открыть файл my\_first\_file.txt и добавить в него новый абзац и закоммитить “изменения от ФИО”(свои данные).



1. Зафиксировать изменения и отправить их на удаленный репозиторий (выполняется всеми студентами подгруппы.



Товарищ коммитит и пушит

1. Просмотреть историю изменен

Изображение выглядит как текст

Автоматически созданное описание

Для выполнения работы необходимо изучить следующий материал:

<https://git-scm.com/book/en/v2>

1. [Getting Started](https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control)
2. [Git Basics](https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository)

### **Список необходимых команд**

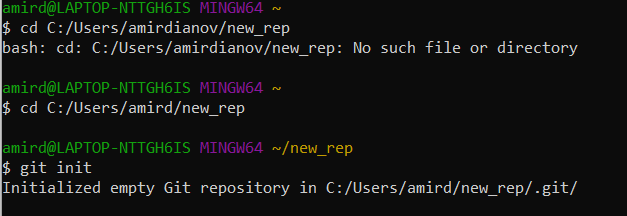
Для выполнения работы потребуются команды:

1. [git init](https://git-scm.com/docs/git-init) — инициализация локального репозитория
2. [git status](https://git-scm.com/docs/git-status) — просмотр состояния файлов в локальном репозитории
3. [git log](https://git-scm.com/docs/git-log) — просмотр истории изменений
4. [git add](https://git-scm.com/docs/git-add) — индексация файлов
5. [git commit](https://git-scm.com/docs/git-commit) — создание коммита
6. [git remote](https://git-scm.com/docs/git-remote) — управление связанными удаленными репозиториями
7. [git push](https://git-scm.com/docs/git-push) — обновление ссылок в удаленном репозитории

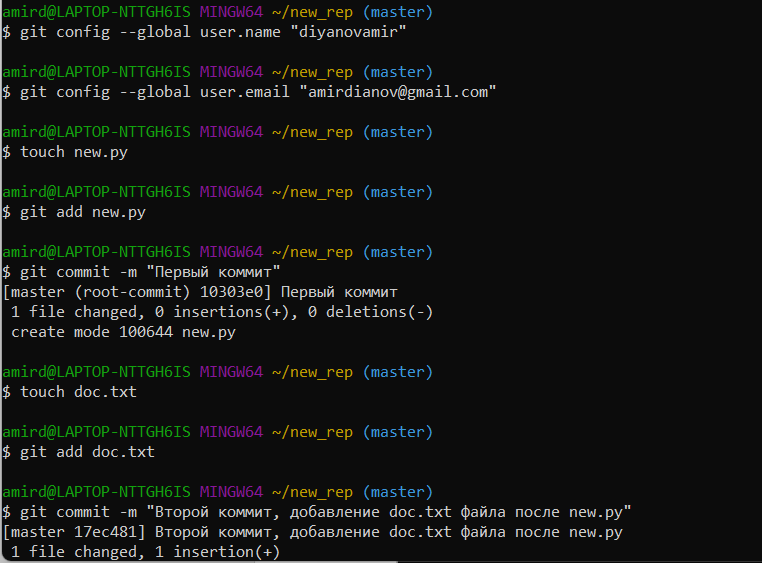
B) Создание веток и коммитов

Ветка в git'е —это просто легковесный подвижный указатель на один из коммитов. Ветка по умолчанию в git'е называется master. Когда происходит создание коммита на начальном этапе, доступна ветка master, указывающая на последний сделанный коммит. При каждом новом коммите она сдвигается вперёд автоматически. Для того чтобы создать новую ветку используется команда gitbranch.Эта команда создаст новый указатель на тот самый коммит,на котором сейчас находится git

1. Создать новый репозиторий branches-basics.

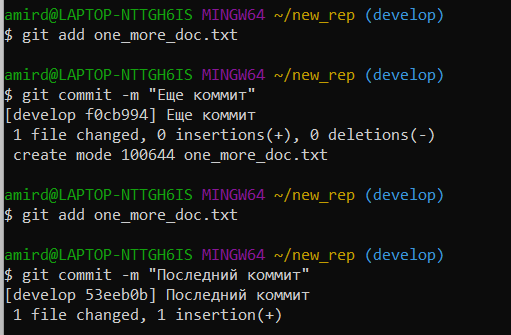


1. В основной ветке master создать несколько коммитов.

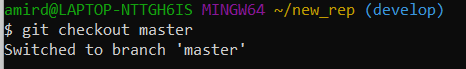


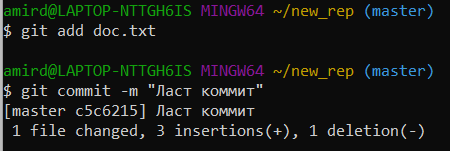
1. Создать новую ветку develop. Выполнить несколько коммитов.\



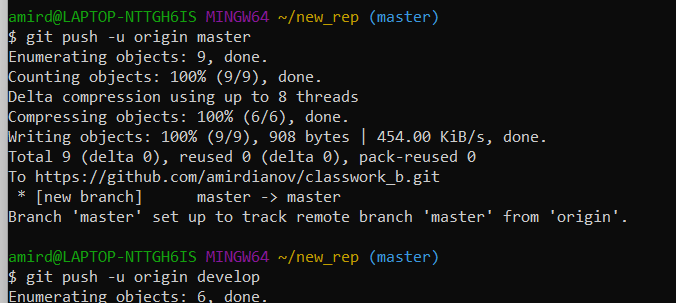


1. Вернуться на ветку master. Создать коммит.





1. Внести изменения в рабочую копию репозитория, не коммитить. Переключиться на ветку develop



1. Предоставить доступ к репозиторию ещё одному пользователю – соседу по парте и попросить закоммитить что-нибудь.

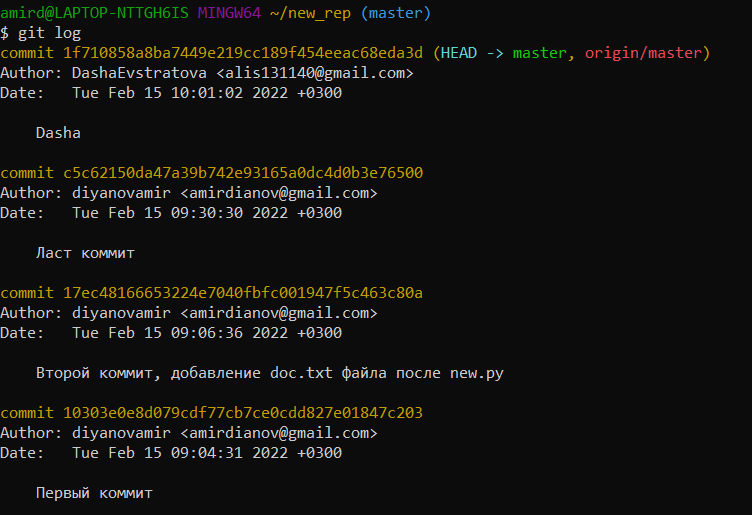
После того как будет создано несколько коммитов, вероятнее всего появится необходимость просмотреть, что же происходило с этим репозиторием. Наиболее простой и в то же время мощный инструмент для этого —команда gitlog.

По умолчанию,без аргументов,gitlog выводит список коммитов созданных в данном репозитории в обратном хронологическом порядке. То есть самые последние коммиты показываются первыми.

Один из наиболее полезных параметров —это -p, который показывает дельту(разницу/diff), привнесенную каждым коммитом. Также можно использовать -2, что ограничит вывод до 2-х последних записей

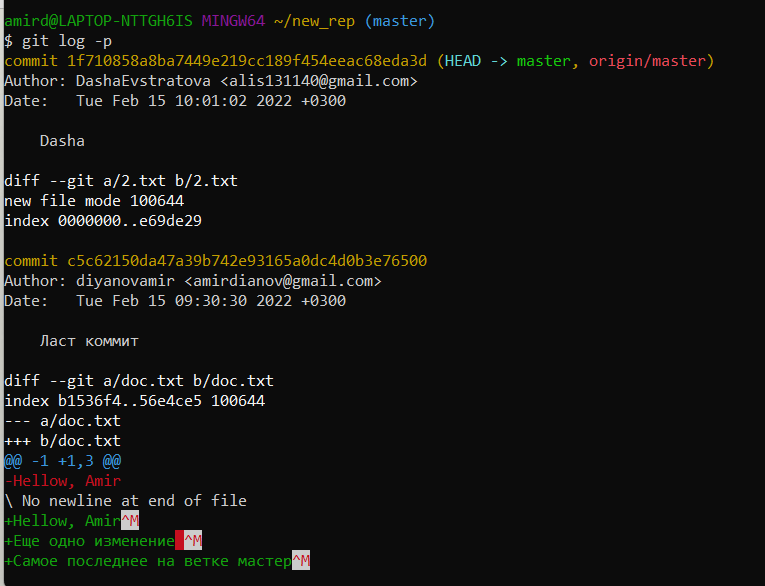
1. Изучить возможности команды gitlog, выполнить различные варианты вывода информации и ее отбора.

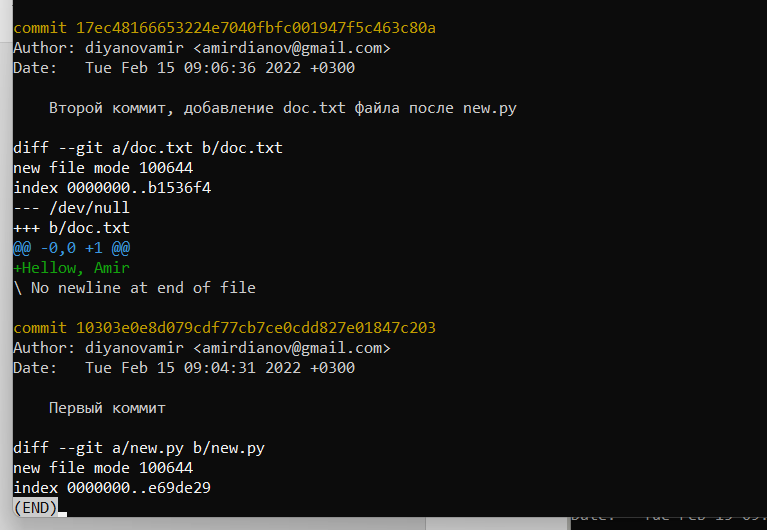
В данном случае команда git log вывод последовательность коммитов по определенной ветке, в данном случае – master  
Вверху находится последний коммит



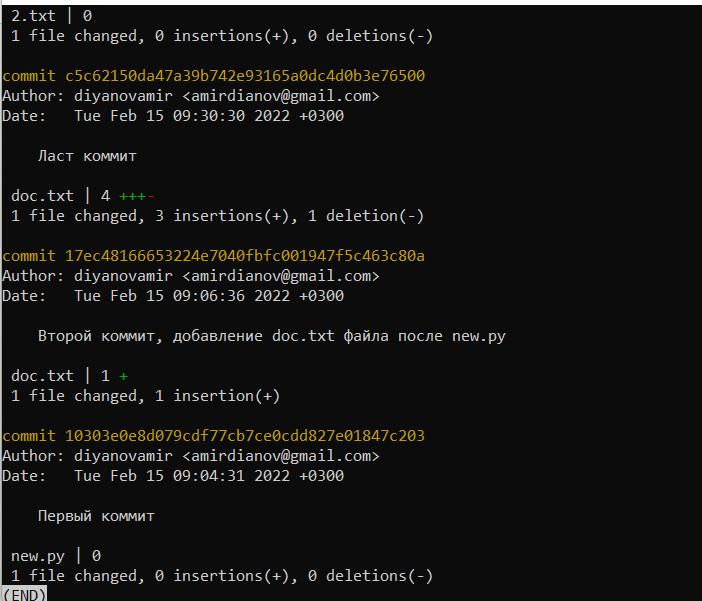
Команда git log -p показывает разницу всех последних коммитов, в данном случае добавление нескольких строчек в текстовый документ

Также командой git log -p -2 мы можем ограничить количество последних коммитов до двух (-3 до трех и тп)

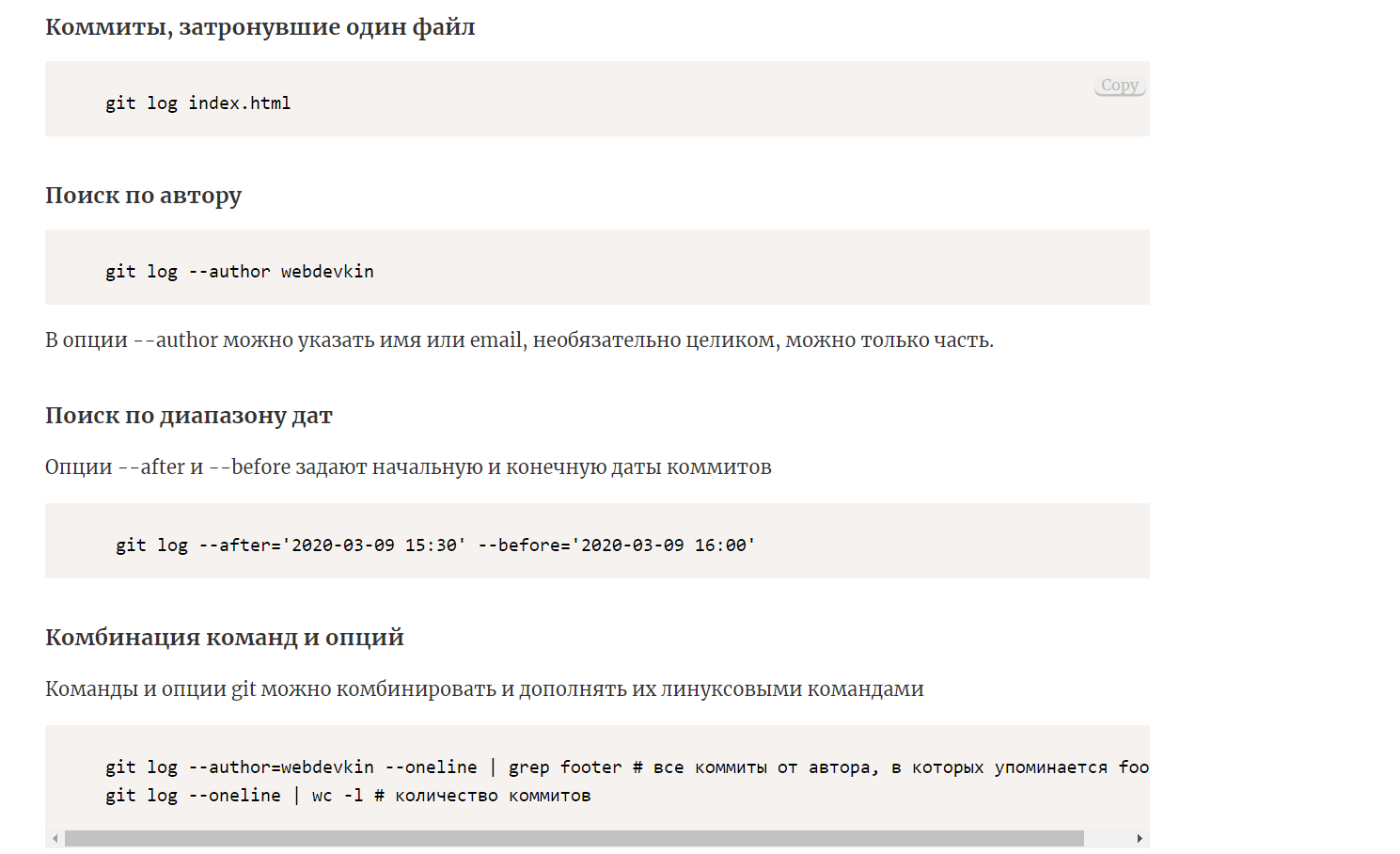


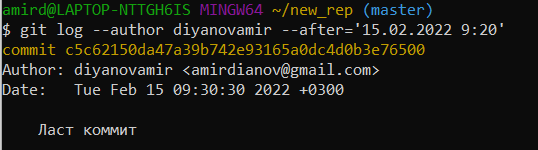


Команда git log –stat выведет где под каждым коммитом написано количество добавленных и удаленных строк



1. Вывести коммиты, автором которых являетесь Вы, за последний месяц.





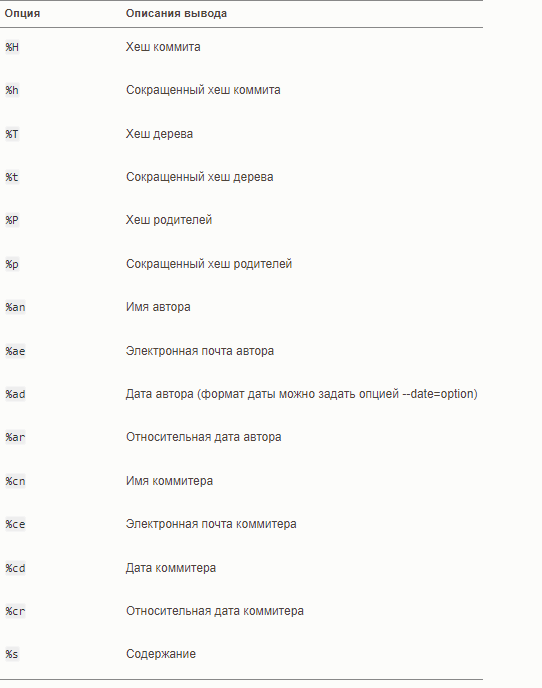
Коммиты моего имени за последние несколько дней (вывел не за месяц, чтобы проверить работоспособность, коммиты максимум 5 дневной давности что ли)

1. Вывести все коммиты в формате: короткий хеш, автор, комментарий.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

1. Вывести все коммиты, в сообщении которых присутствует какое-нибудь слово

Изображение выглядит как текст

Автоматически созданное описание

1. Вывести все коммиты за текущий месяц с информацией о том, какие файлы были изменены

Изображение выглядит как текст

Автоматически созданное описание

С) Продолжение

1. Создать локальный репозиторий.

2.Создать в локальном репозитории приложение , выводящее в меню выбора, состоящее из следующих пунктов:

-ввести А

-ввести В

-выполнить операцию

“+”-выполнить операцию “-

”-выполнить операцию

“\*”-выполнить операцию “/”

3.Синхронизировать локальный репозиторий, с созданным на GitHub.

4.Создать ветви для реализации функционала каждого из пунктов меню в разных директориях.

Изображение выглядит как текст, монитор, снимок экрана, экран

Автоматически созданное описание

5.Реализовать функции меню, каждую в своей ветви.

6.Синхронизировать локальные репозитории через репозиторий на GitHub.

7.Создать ветвь “final”и объединить в ней все ветви проекта.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Возникает подобного рода конфликт при слиянии, для этого нужно вернуться в файл, редактировать его в ручную, далее выполнить git add = для гита, что конфликт исчерпан, и коммитим

8.Объединить ветви “final”и “master”.

Изображение выглядит как текст

Автоматически созданное описание

9.Синхронизировать финальную версию проекта с репозиторием на GitHub.

Источники

[Branches in a Nutshell](https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell)

Команды

* [git checkout](https://git-scm.com/docs/git-checkout)
* [git branch](https://git-scm.com/docs/git-branch)

D) Прятанье

Часто возникает такая ситуация, что пока идет работа над частью своего проекта, всё находится в беспорядочном состоянии, а нужно переключить ветки, чтобы немного поработать над чем-то другим. Проблема в том, что делать коммит с наполовину доделанной работой только для того, чтобы позже можно было вернуться вэто же состояние не хотелось бы. Ответ на эту проблему —команда gitstash. Прятанье поглощает грязное состояние рабочего каталога,то есть изменённые отслеживаемые файлы и изменения в индексе, и сохраняет их в стек незавершённых изменений, которые потом в любое время можно снова применить.

1. Проверить какие ветки слиты с веткой master, а какие нет.

2.Создать новую ветку work и перейти в нее.

3.Изменить файл 1.txt.7.Спрятать данные изменения.

4.Развернуть обратно данные изменения с опцией --index.

5.Развернуть спрятанные изменения в новую ветку.

E) Зачастую, имеется группа файлов, которые не только нет необходимости автоматически добавлять в репозиторий, но и видеть в списках неотслеживаемых. К таким файлам обычно относятся автоматически генерируемые файлы (различные логи, результаты сборки программ и т.п.). В таком случае, необходимо создать файл .gitignore с перечислением шаблонов соответствующих таким файлам. шаблонам в файле .gitignore применяются следующие правила:

●Пустые строки, а также строки, начинающиеся с # (символ коммента-рия), игнорируются.

●Можно использовать стандартные glob шаблоны.

●Можно заканчивать шаблон символом слэша (/) для указания каталога.

●Можно инвертировать шаблон, использовав восклицательный знак (!) в качестве первого символа.Glob-шаблоны представляют собой упрощённые регулярные выражения ис-пользуемые командными интерпретаторами. Символ \*соответствует 0 или более символам; последовательность [abc]—любому символу из указанных в скобках (в данном примере a, b или c); знак вопроса (?) соответствует одному символу; [0-9] соответствует любому символу из интервала (в данном случае от 0 до 9)

1. Создать дополнительные папку temp и файлы с расширением log.
2. Создать файл .gitignore и добавить в игнорирование папку temp и файлы c расширением log.
3. Закоммитить добавление файла .gitignore
4. Внести изменения в файл 1.txt,добавив строчку “row to index”, проиндексировать данные изменения. Еще раз внести изменения в файл, добавив строчку “row no index”
5. Посмотреть индексированные и неиндексированные изменения исполь-зуя команду git diff.
6. Удалить файл 1.txt, зафиксировать данное удаление.

F) Одна из типичных отмен происходит тогда, когда коммит сделан слишком ра-но, например,не были добавлены какие-либо файлы, или перепутан комментарий к коммиту. Если необходимо сделать этот коммит ещё раз, можно выполнить git commitс опцией –amend. Эта команда берёт индекс и использует его для коммита. Если после послед-него коммита не было никаких изменений (например, приведенная команда была запущена сразу после предыдущего коммита), то состояние проекта будет абсолют-но таким же и всё, что изменится, это комментарий к коммиту.Git использует два основных типа меток: легковесные и аннотированные. Легковесная метка —это что-то весьма похожее на ветку, которая не меняется —это просто указатель на определённый коммит. А вот аннотированные метки хранятся в базе данных Git'а как полноценные объекты. Они имеют контрольную сумму, со-держат имя поставившего метку, e-mail и дату, имеют комментарий и могут быть подписаныи проверены с помощью GNU Privacy Guard (GPG). Обычно рекомендуется создавать аннотированные метки, чтобы иметь всю перечисленную информацию; но если необходимо сделать временную метку или по какой-то причине нет необходимости сохранять остальную информацию, то для этого годятся и легковесные метки

1.Создать три файла: 1.txt, 2.txt, 3.txt.

2.Проиндексировать первый файл и сделать коммит с комментарием “add 1.txt file”.

5.Проиндексировать второй и третий файлы.

6.Удалить из индекса второй файл.

7.Перезаписать уже сделанный коммит с новым комментарием “add 1.txt and 3.txt”

8.Создать аннотированную метку с названием v0.01.9.

Создать легковесную ветку указывающую на первый коммит в репозитории.

G) Разобрать, что такое конфликты. Дать здесь описание

1. Создать конфликт.
2. Посмотреть в каких файлах есть конфликты.
3. Устранить конфликты.

H) Смоделировать групповую работу над одним проектом с группой из 3-4 человек. Продемонстрировать.

Контрольные вопросы

1. Что такое система управления версиями?
2. Какие проблемы решает Git?
3. Что такое репозиторий. Как создать репозиторий?
4. Что такое GitBash?
5. Для чего нужна команда ls
6. .Как сменить директорию?
7. Как отобразить текущую директорию?
8. Для чего задаются настройки Git?
9. Как применить одинаковые настройки для всех репозиториев?
10. Что такое ветка. Как создать ветку?
11. Как провести слияние?
12. Как разрешить конфликт и что это такое?
13. Как провести откат? Различия в reset и revert, мягкий и жесткий reset
14. Какова последовательность действий при работе с локальным репозиторием?
15. Какова последовательность действий при работе с удаленным репозиторием?
16. Каковы возможности при работе с удаленным репозиторием? Как его клонировать,

получать и отправлять данные?

1. Как отобразить текущий статус репозитория?
2. .Какие файлы называются Untracked files?
3. Как отменить коммит и почему это делать не рекомендуется?