Практическое задание 9. Git

На оценку 5 необходимо сделать все, на оценку 4 выполнить 3 и 4.

Начальная настройка

Создать репозиторий, где хранятся наши лабы на Github.com, клонируем его к себе. Если репозиторий новый – инициализируем его командами из подсказки Github.com после создания репозитория.

Создать в репо папку для лабы с git'ом **– все действия будем делать внутри этой папки, например "lab9_git"**

Создать текстовый файл README.md (markdown формат https://www.markdownguide.org/getting-started/) куда будете документировать скриншоты и краткое описание о каждом проделанном шаге. Плюсом будет если некоторые скриншоты будут подкреплены парой объяснительных слов. Этот текстовый файл должен быть запушен на ваш удалённый репо.

Логинимся:

1. git config --global user.name "John Doe"
2. git config --global user.email "johndoe@example.com

Выбираем текстовый редактор (если не слышали про vim или он вас пугает):

- git config --global core.editor nano

Windows:

- git config --global core.editor notepad

Полезные команды:

- git add
- git commit
- git commit -m "My commit message"
- git log
- git log -n 5
- git log --oneline
- git log --oneline –graph
- git log --oneline –graph --all

- git switch
- git switch -c
- git branch
- git branch <branch-name>
- git branch -v
- git branch -d <branch-name>
- git diff
- git diff <branchA> <branchB>
- git checkout
- git checkout -b
- git commit -m “my short commit message”
- git restore –staged
- git merge
- git mergetool --tool=vimdiff

Из длинных git комманд можно делать алиасы (псевдоним, сокращение), например:

git config --global alias.lol 'log --oneline --graph --all' добавит алиас

git lol (что сокращенно тоже что и git log --oneline --graph --all)

Оценка 3:

1. Используйте git status, чтобы узнать, на какой ветке вы находитесь.
2. Как выглядит git log?
3. Создайте файл sort.c и вставьте туда код функции любой сортировки (**только ф-ия сортировки**)
4. Как сейчас выглядит вывод git status?
5. Добавьте файл в область stage (add)
6. Как сейчас выглядит git status?
7. Закоммитить файл в репозиторий
8. Как сейчас выглядит git status?
9. Добавить комментарий с любым текстом в этот же файл
10. Как сейчас выглядит git status?
11. Добавьте (add) изменение файла
12. Как сейчас выглядит git status?
13. Измените файл еще раз (можно добавить еще комментарий или убрать старый)
14. Сделайте коммит
15. Как сейчас выглядит status? Журнал (log)?
16. Добавьте в stage и закоммитьте последнее изменение

17. Запушим на удаленный репо (git push)

Теперь мы немного поиграем с ветками.

1. Используйте git branch mybranch (или git checkout -b mybranch), чтобы создать новую ветку с именем mybranch.
2. Снова используйте git branch, чтобы увидеть новую созданную ветку.
3. Используйте git switch mybranch (или git checkout mybranch), чтобы переключиться на новую ветку.
4. Как изменяется вывод git status при переключении между master и новой веткой, которую вы создали?
5. Убедитесь, что вы находитесь на своей ветке mybranch, прежде чем продолжить.
6. Создайте файл с именем file1.txt и своим именем.
7. Добавьте файл и закоммитьте это изменение.
8. Используйте git log --oneline --graph, чтобы увидеть, что ваша ветка указывает на новый коммит.
9. Вернитесь к ветке с именем master.
10. Используйте git log --oneline --graph, что изменилось?
11. Создайте новый файл с именем file2.txt и закоммитьте его.
12. Используйте git log --oneline --graph --all, чтобы увидеть, что ваша ветка указывает на новый коммит, и что теперь у двух веток разные коммиты.
13. Переключитесь на вашу ветку mybranch.
14. Наш file2.txt пропал?
15. Используйте git diff mybranch master, чтобы увидеть разницу между двумя ветками.
16. Добавить текстовый документ со скриншотами в ветку **mybranch**. Закоммитить и запушить на удаленный репо ветку mybranch (git push -u origin mybranch)
17. Убедиться, что в github.com две ветки master и mybranch. Не забыть запушить изменения master ветки в master

Оценка 4:

Выполнить на оценку 3, а затем..

1. Переключитесь на ветку mybranch. В ней будет файл sort.c из предыдущих шагов с функцией сортировки
2. Перезапишите содержимое в sort.c добавив функцию main(), в которой будет объявлен массив из нескольких чисел (пример int a[] = {4, 2, 0};) и вызвана функция сортировки для этого массива.
3. Что вам говорит git diff?
4. Что вам говорит git diff --staged? Пустой?

5. Добавьте в staged файл sort.c
6. Что вам говорит git diff?
7. Что вам говорит git diff --staged?
8. Удалите любое из чисел в массиве в sort.c:
9. Что вам говорит git diff?
10. Что вам говорит git diff --staged?
11. Объясните, что происходит
12. Запустите git status и обратите внимание, что sort.c присутствует дважды в выводе.
13. Запустите git restore --staged sort.c, чтобы отменить индексацию изменения
14. Что вам теперь говорит git status?
15. Индексируйте изменение (add) и сделайте коммит
16. Как выглядит журнал?
17. Добавьте в sort.c в main() printf(“hello git\n”);.
18. Каково содержимое sort.c?
19. Что нам говорит git status?
20. Запустите git restore sort.c
21. Каково содержимое sort.c?
22. Что нам говорит git status?
23. Запушить на удаленный репо ветку.

Теперь поиграемся с ветками и ff-merge
Помните: когда вы хотите обновить ветку так, чтобы она также имела все изменения из другой ветки, используйте команду 'git merge [имя ветки]', где [имя ветки] - ветка, из которой мы хотим смержить наши изменения.

1. Создать файл greeting.txt, проиндексировать его и закоммитить с сообщением “Add file greeting.txt”.
2. Добавить в этот файл слово hello, индексируем и коммитим с текстом "Add content to greeting.txt"
3. Создайте ветку с именем feature/uppercase (да, feature/uppercase — это совершенно допустимое имя ветки и общепринятое соглашение).
4. Переключитесь на эту ветку
5. Каков вывод git status?
6. Отредактируйте greeting.txt, чтобы он содержал приветствие в верхнем регистре (HELLO)
7. Добавьте файл greeting.txt и закоммитьте
8. Каков вывод git branch?
9. Каков вывод git log --oneline --graph –all
10. Переключитесь на главную ветку
11. Используйте cat, чтобы увидеть содержимое файла greetings.txt
12. Сравните ветки

13. Объедините ветки
14. Используйте cat, чтобы увидеть содержимое файла greetings.txt
15. Удалите ветку с заглавными буквами (feature/uppercase)
16. Смержить ветку mybranch в master (git merge)
17. Что выводит git log --oneline --graph –all?
18. Запушить изменения ветки master на удаленный репо.
19. Запушить документ с результатами вашей работы

Оценка 5:

Выполнить на оценку 4, а затем попробуем исправить пару merge conflict'ов

1) Создать ветку branch1, переключиться на нее
2) Выполнить команду
   $ echo "This is a relevant fact" > file.txt
3) Закоммитить это изменение
4) Переключиться на главную ветку и выполнить команду
   echo "This is an indispensable truth!" > file.txt
5) Закоммитить изменения в master
6) Каков вывод git log --oneline --graph –all
7) Использовать команду git merge чтобы смержить ветку branch1 в master (получим конфликт это норм)
8) Что показывает git status?
9) Посмотреть содержимое файла file.txt и в любом любимом текстовом редакторе исправьте конфликт. Желательно делать это в VSCode или любой другой умной среде, но если нет vscode, то можно через команду git mergetool.
10) Что показывает git log --oneline –graph?
11) Запушить изменения

Починим Merge конфликты для сортировки MergeSort на python.
Содержимое base.py, lefty.py и righty.py находится в конце этого документа.

1) Находясь в ветке master создадим файл mergesort.py с содержимым из **base.py** (**см. приложение**)
2) Проиндексируем файл и закоммитим
3) Создадим новую ветку Mergesort-Impl и переключимся на нее.
4) Содержимое файла mergesort.py заменим на код из righty.py
5) Коммитим изменения.
6) Переключаемся на master и меняем все содержимое mergesort.py на lefty.py

7) Коммитим изменения.
8) Что показывает git log --oneline –graph?
9) Что показывает git branch?
10) Необходимо смержить Mergesort-Impl в master.
11) После исправления всех merge конфликтов запушить в master изменения.
12) Запушить документ с результатами вашей работы

Под Звездочкой (+ балл):

Создать свой сабмодуль и добавить его к себе в репо.

Содержимое сабмодуля **может быть любым**, например какая-то ваша библиотека с кодом или репо с разными функциями (сортировки или списки или файл-отчет по проделанной работе за эту лабу..). То есть у вас будет репо с файлами и он же будет сабмодуль в вашем основном репо с лабами.

Показать инициализацию сабмодуля: файлов сабмодуля не было, а сделали init и файлы появились.

Приложение на оценку 5 (base.py, righty.py, lefty.py):

base.py:

```
def merge_sort(m):
        """Sort list m, using merge sort"""
        return m.sort() # TODO: Replace with actual implementation
```

righty.py:

```
from heapq import merge

def merge_sort2(m):
    """Sort list, using two part merge sort"""
    if len(m) <= 1:
        return m

    # Determine the pivot point
    middle = len(m) // 2

    # Split the list at the pivot
    right = m[middle:]
    left = m[:middle]
```

```
    # Sort recursively
    right = merge_sort2(right)
    left = merge_sort2(left)

    # Merge and return
    return list(merge(right, left))

def merge_sort4(m):
    """Sort list, using four part merge sort"""
    if len(m) <= 4:
        return sorted(m)

    # Determine the pivot point
    middle = len(m) // 2
    leftMiddle = middle // 2
    rightMiddle = middle + leftMiddle

    # Split the list at the pivots
    first = m[:leftMiddle]
    second = m[leftMiddle:middle]
    third = m[middle:rightMiddle]
    fourth = m[rightMiddle:]

    # Sort recursively
    first = merge_sort4(first)
    second = merge_sort4(second)
    third = merge_sort4(third)
    fourth = merge_sort4(fourth)

    # Merge and return
    return list(merge(first,second, third, fourth))
```

lefty.py:

```
from heapq import merge

def merge_sort2(m):
    """Sort list, using two part merge sort"""
    if len(m) <= 1:
        return m

    # Determine the pivot point
    middle = len(m) // 2

    # Split the list at the pivot
```

```
    left = m[:middle]
    right = m[middle:]

    # Sort recursively
    right = merge_sort2(right)
    left = merge_sort2(left)

    # Merge and return
    return list(merge(right, left))

def merge_sort4(m):
    """Sort list, using four part merge sort"""
    if len(m) <= 4:
        return sorted(m)

    # Determine the pivot point
    middle = len(m) // 2
    leftMiddle = middle // 2
    rightMiddle = middle + leftMiddle

    # Split the list at the pivots
    first = m[:leftMiddle]
    second = m[leftMiddle:middle]
    third = m[middle:rightMiddle]
    last = m[rightMiddle:]

    # Sort recursively
    first = merge_sort4(first)
    second = merge_sort4(second)
    third = merge_sort4(third)
    last = merge_sort4(last)

    # Merge and return
    return list(merge(first, second, third, last))
```