Семинар #3: git, продвинутый. Практика.

Как сдавать задачи

Для сдачи ДЗ вам нужно создать репозиторий на GitLab (если он ещё не создан) под названием `devtools-homework`. Структура репозитория должна иметь вид:

```
├── seminar3_advanced_git/
│   ├── 01.txt
│   ├── 02.sh
│   ├── 03.sh
│   └── ...
└── ...
```

Если не указано что-то другое, то для каждой задачи нужно будет создать текстовый файлы с расширением `.txt` или `.sh`, в котором нужно написать команды, которые использовались при решении данной задачи.

Задача 1. Работа с репозиторием

Как сдать эту задачу

Создайте файл `01.txt` и записывайте в него команды, которые нужно выполнить для той или иной подзадачи. В некоторых задачах могут возникнуть конфликты слияния, которые необходимо исправлять вручную. Чтобы указать, что возник конфликт оставьте комментарий `"Конфликт, разрешаю вручную"`. После некоторых команд может открыться текстовый редактор и попросить что-то ввести. Чтобы отметить это в ответе напишите комментарии, содержащие следующие строки: `"Открылся редактор, ввожу:"` и дальше то, что вы вводите в редактор. Выглядеть это должно примерно так:

```
# Подзадача d
git merge bob
# Конфликт, разрешаю вручную
git add имена файлов
git merge --continue
# Открылся редактор, ввожу:
# This is my merge commit!

# Подзадача e
git reset --hard main~

# Подзадача f
```

В некоторых задачах может понадобиться обратиться к `git reflog`. Чтобы указать это создайте комментарий, в котором напишите следующее: `"Открываю reflog и ищу нужный коммит"`, примерно вот так:

```
git reset --hard HEAD~2
# Открываю reflog и ищу нужный коммит
git reset --hard 24e1f51
```

Подготовка. Генерация репозитория, который будет использоваться в подзадачах

Для решения этой задачи вам потребуется создать репозиторий коммитов. Его можно найти по адресу: mipt-hsse.gitlab.yandexcloud.net/v.biryukov/utils. Клонируйте этот репозиторий себе:

```
git clone git@mipt-hsse.gitlab.yandexcloud.net:v.biryukov/utils.git
cd utils
```

Подзадачи

a. **Просмотр содержимого репозитория**
Перейдите на все ветки и просмотрите, какие файлы содержатся в их последних коммитах:

```
$ git switch alice
$ git switch bob
$ git switch casper
$ git switch main
```

Это важно сделать ещё и потому что при клонировании репозитория создаются только remote-ветки (вроде `origin/alice`), а обычные локальные ветки (вроде `alice`) не создаются. Их можно создать вручную или они создаются автоматически при переходе на них.

b. **Просмотрите граф коммитов**
Это можно сделать на GitLab во вкладке `Code -> Repository Graph` или у вас на машине, с помощью команды:

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

Для вывода на экран, если вывод не помещается на экран полностью, команда `git log` используется программу `less`. Чтобы показать следующую строчку используйте `Enter`. Чтобы выйти из программы `less` просто нажмите клавишу `Q`.

c. **Новый alias**
Добавьте новый `alias` для этой команды, назовите его `lg`. То есть теперь, при вызове:

```
$ git lg
```

Должно печататься то же, что и при вызове

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

d. **Слияние с конфликтом**
Слейте ветку `bob` на ветку `main`. Для этого вам нужно перейти в ветку `main` и вызвать:

```
$ git merge bob
```

При этом могут возникнуть конфликты. Разрешите их. (в этой и последующих задачах необязательно, чтобы код после слияния был корректным, так как это задание на git, а не на язык программирования). Посмотрите как выглядит граф коммитов после слияния с помощью `git lg`.

e. **Отмена слияния**
Отмените только что выполненное слияние.

f. **Перебазирование с конфликтом**
Перебазируйте ветку `bob` на ветку `main`. Для этого вам нужно перейти в ветку `bob` и вызвать:

```
$ git rebase main
```

При этом могут возникнуть конфликты. Разрешите их. Посмотрите как выглядит граф коммитов после перебазирования с помощью `git lg`.

g. **Отмена перебазирования**
Отмените только что выполненное перебазирование.

h. **Копия коммита с конфликтом**
Перейдите на ветку `main` и скопируйте (`git cherry-pick`) в неё один коммит с хэшем `893e28d` из ветки `bob`. При этом могут возникнуть конфликты. Разрешите их. Посмотрите как выглядит граф коммитов с помощью `git lg`.

i. **Отмена `cherry-pick`**
Отмените только что сделанный `cherry-pick`.

j. **Слияние с помощью перемотки**
Перейдите на ветку `alice` и слейте в ветку `alice` ветку `main`, используя *перемотку* (*fast-forward merge*). Посмотрите как выглядит граф коммитов с помощью `git lg`.

k. **Отмена слияния перемоткой**
Отмените только что выполненное слияние перемоткой.

l. **Слияние без быстрой перемоткой**
Перейдите на ветку `alice` и слейте в ветку `alice` ветку `main`, без перемотки. Посмотрите как выглядит граф коммитов с помощью `git lg`.

m. **Отмена последнего слияния**
Отмените только что выполненное слияние.

n. **Просмотр файлов**
Напишите команды, которые печатают файлы на экран:

- файл `arithmetic.py` из ветки `main`
- файл `arithmetic.py` из ветки `casper`
- файл `arithmetic.py` из коммита `casper~5`

o. **Разница**
Напишите команды, которые печатают на экран разницу (`diff`) между следующими коммитами или отдельными файлами:

- между ветками `main` и `casper`
- между коммитами `casper` и `casper~5`
- между файлом `integration.py` в ветке `main` и файлом `integration.py` ветки `casper`.
- между файлом `misc.py` коммита `fe2e890` и файлом `arithmetic.py` коммита `765df07`.

p. **Патч-файл**
Создайте патч-файл, содержащий `diff` последних двух коммитов ветки `casper`. Перейдите в ветку `main` и примените этот патч с помощью команды `git apply`. При этом могут возникнуть конфликты. Используйте команду `git apply` с опцией `--3way` для разрешения конфликтов. Закоммите изменения, созданные патчем, в ветке `main`. Сам патч-файл удалите.

q. **git bisect**
Перейдите в ветку `casper` и запустите скрипт `sorting.py`

```
python ./sorting.py
```

Вы увидите, что одна из сортировок работает неправильно, хотя в других ветках эта сортировка работала правильно. Значит в одном из коммитов ветки `casper` была допущена ошибка. Найдите коммит, в котором была допущена ошибка с помощью `git bisect`.
Для сдачи этой задачи укажите в файле `01.txt` коммит, файл и строчку в которой впервые возникает ошибка.

r. **Интерактивный `rebase`**
Сделайте интерактивный `rebase` ветки `casper` с момента отсоединения её от других веток. Склейте вместе коммиты ветки `casper`, так чтобы там осталось 4 коммита. Выберете коммиты так, чтобы они были примерно равномерно распределены по истории ветки `casper`.

s. **Отмена интерактивного `rebase`**
Отмените только что сделанный интерактивный `rebase`.

t. **git add -p**
Перейдите на ветку `casper` и добавьте изменения. В рабочей папке добавьте комментарии к функциям `add`, `factorial`, `is_prime` и `is_perfect_number` из файла `arithmetic.py`. Комментарии должны выглядеть примерно так:

```
# This function adds two numbers
def add(a: float, b: float) -> float:
    return a + b
```

Добавьте в индекс только изменения, связанные с функциями `add` и `is_prime`. Остальные изменения добавлять не нужно. Добавлять новый коммит не нужно. Используйте команду `git add` с опцией `-p`.

u. **git stash**
После изменений, сделанных в прошлой подзадаче, попробуйте переключиться на ветку `main`. Произойдёт ошибка, так как есть изменения в рабочей директории и в индексе. Используйте `git stash`, чтобы спрятать изменения. Перейдите на ветку `main` и сделайте там какой-нибудь коммит на ваш выбор. Вернитесь на ветку `casper` и вытащите изменения из `stash`. Вытащить изменения нужно таким образом, чтобы те изменения, которые были в индексе, остались в индексе, а те изменения, которые были в рабочей папке, остались в рабочей папке.

v. **Перенос git stash**
Снова добавьте эти же изменения в `stash`. Перейдите на ветку `bob` и вытащите изменения, сделанные на ветке `casper`, в ветку `bob`. Возможен конфликт. Исправьте конфликт и сделайте коммит с этими изменениями в ветке `bob`.

w. **Фильтрация репозитория**
`git filter-repo` это не команда самого git, а сторонняя программа. Для решения этой подзадачи необходимо установить её. Перед выполнением этой подзадачи на всякий случай скопируйте весь репозиторий.

Напишите команды, используя `git filter-repo`, которые делают следующее:

- переименовывают файл `arithmetic.py` в файл `ar.py` во всех коммитах репозитория
- удаляют файл `sorting.py` во всех коммитах репозитория

Задача 2. `autocrlf`

Задача 3. `.gitattributes`

Задача 4. Хуки git

Напишите pre-commit хук, который будет проверять, что все файлы имеют расширение `.txt`. Проверьте работу этого хука, закоммитив файл с другим расширением.

Задача 5. Низкоуровневый git

В этом задании нельзя использовать команды, которые мы изучали прежде (кроме `git init`). Можно использовать только следующие низкоуровневые команды:

- git init
- git hash-object
- git cat-file
- git ls-files
- git mktree
- git commit-tree
- git update-ref
- git ls-tree

А также команды для проверки результатов:

- git status
- git log

- git show
- git diff

Создайте репозиторий, состоящий из 3-х коммитов и 2-х веток, используя только эти команды.

Задача 6. Ещё более низкоуровневый git

Решите предыдущую задачу вообще без использования команд git. Можно только использовать команды для проверки результатов:

- git status
- git log
- git show
- git diff