**Рекомендации по написанию коммитов**

Во всех отправляемых вами фрагментах кода не должно быть ошибок, связанных с пробелами. Проверить их наличие можно командой

$ git diff --check

Старайтесь, чтобы каждый коммит представлял собой логически обособленный набор изменений. Например, если вы решили 5 различных вопросов сразу, не стоит отправлять один огромный коммит – следует разбить изменения на 5 коммитов, по одному на проблему, с содержательным сообщением фиксации в каждом случае. Если все изменения относятся к одному файлу, его можно индексировать по частям:

$ git add --patch <file>

Написание содержательных сообщений можно упростить, следуя определённым рекомендациям по форматированию: как правило, сообщение должно начинаться со строки с лаконичным описанием изменений, содержащей не более 50 символов. Затем следует пустая строка и более подробное описание (~72 символа).

В подробный текст рекомендуется включить информацию о причине изменений и сопоставление вашей реализации с предыдущим поведением.

Пример шаблона сообщения:

Краткое (до 50 символов) описание изменений  
  
Более детальный поясняющий текст, если он требуется. В некоторых контекстах первая строка рассматривается как тема электронного письма, а остальной текст – как тело письма. Важна пустая строка, отделяющая тему от тела (если тело присутствует); в противном случае некоторые инструменты могут воспринять написанное некорректно.  
  
Дальнейшие абзацы идут после пустых строк.  
  
 - Возможны маркированные списки.  
  
 \* Роль маркера обычно играет дефис или звёздочка, с одним пробелом перед ними и пустыми строками между пунктами.

**Работа в маленькой группе**

На данном практическом занятии вы начнёте отрабатывать работу простейшей организационной структуры – закрытый (недоступный остальному миру) проект с одним или двумя разработчиками.

Такая среда позволяет проводить рабочий процесс, схожий с централизованными системами контроля версий, основное отличие будет в том, что при фиксации изменений слияние проходит на стороне клиента, а не на сервере, как в CVCS

Рассмотрим пример двух разработчиков А и В  
Разработчик А клонирует репозиторий, добавляет свои изменения и фиксирует их локально.

# Компьютер А  
$ git clone [A@githost:simplegit.git](mailto:A@githost:simplegit.git)  
Initialized empty Git repository in /home/A/simplegit/.git/  
$ cd simplegit/  
$ nano example.txt  
$ git commit –am 'added another example'

Второй разработчик тоже клонирует репозиторий и фиксирует внесённые в копию изменения

# Компьютер В  
$ git clone [B@githost:simplegit.git](mailto:B@githost:simplegit.git)   
Initialized empty Git repository in /home/B/simplegit/.git/  
$ cd simplegit  
$ nano TODO  
$ git commit –am 'add reset task'

Затем отправляет свою работу на сервер:

# Компьютер В  
$ git push origin master

Разработчик А также пытается отправить свои изменения на сервер:

# Компьютер А  
$ git push origin master  
! [rejected] master -> master (non-fast forward)  
error: failed to push some refs to '[A@githost:simplegit.git](mailto:A@githost:simplegit.git)'

А получает отказ в совершении операции, так как В успел отправить свои изменения первым. Это важный момент совместной работы в Git.

Мы видим, что разработчики редактировали разные файлы; слияние тривиально, но при работе в Git это нужно выполнять вручную. А следует скачать изменения, внесённые В, и выполнить их слияние, и только после этого он получит разрешение на отправку своих изменений:

# Компьютер А  
$ git fetch origin

У А теперь есть ссылка на выложенные В изменения, и он должен осуществить слияние со своей работой:

$ git merge origin/master

Слияние проходит без проблем, и теперь можно отправлять изменения:

$ git push origin master

В это время В работал над иной веткой. Он создал ветку issue54 и выполнил в неё три коммита. Чтобы синхронизировать свою работу с наработками А, В скачивает с сервера изменения:

# Компьютер В  
$ git fetch origin

В считает свою ветку законченной и хочет узнать, с чем нужно слить его работу, чтобы получить возможность отправить её на сервер. Для этого использует git log:

$ git log --no-merges issue54..origin/master  
commit 12s33asd341..  
Author: A <a@example.com>  
Date: Fri May 29 16:01:27 2024 -0700

add reset task

Запись issue54..origin/master служит фильтром, указывающим, что отображать нужно только те коммиты из последней ветки (origin/master), которых нет в первой ветке (issue54)

Для того, чтобы отправить свои изменения на сервер, В необходимо слить в свою локальную ветку master удалённую ветку origin/master, затем слить в master ветку issue54 а потом отправить все изменения на сервер.

Сперва нужно перейти в ветку master:

$ git checkout master

Теперь можно слить ветку origin/master или issue54 – обе они находятся выше по истории коммитов, поэтому порядок не играет особой роли (итоговое состояние будет одинаковым)

$ git merge issue54

$ git merge origin/master

$ git push origin master

Это одна из простейших рабочих схем. Некоторое время вы работаете в тематической ветке, и, когда приходит время, сливаете результаты своего труда в ветку master. Решив, что пришло время поделиться своими наработками с коллегами, вы скачиваете данные с сервера, и если там появились изменения, сливаете к себе ветку origin/master, после чего содержимое master можно отправлять на сервер

**Задача:**

В рамках данного практического занятия необходимо, работая вдвоём, реализовать одно из предложенных приложений, используя IDE PyCharm / VsCode и возможности совместной разработки.

1. **Разбиться на пары.** 
   1. Желательно выбрать в напарники человека, с которым планируете выполнять домашнюю работу к концу семестра.
2. **Подготовить репозитории**
   1. Создать репозиторий на локальной машине в папке проекта pz6  
      $ git init
   2. Подключить удалённый репозиторий к локальному репозиторию  
      $ git remote add origin <repo>
3. **Выбор задачи и удалённой ветки**
   1. Выбираете задачу из перечня и приступаете к её выполнению
   2. При пуше на ваш удалённый репозиторий имя вашей ветки на нём должно состоять из ваших фамилий или первых букв ваших фамилий. Пример:  
      Иванов и Петров -> iva\_pet или ivanov\_petrov  
      $ git push origin Ivanov\_petrov
4. **Совместная работа над проектом**  
   Для успешной сдачи занятия вам обязательно необходимо проработать параллельную разработку, получение и слияние изменений и их передачу, и уметь показать это в истории вашего репозитория (в данном случае ветки на удалённом репозитории). Ход работы должен выглядеть так:  
   1. Обсудить выбранную задачу, способы её решения и что необходимо сделать.
   2. Разделить обязанности. Например, один разработчик отвечает за код, обрабатывающий входные аргументы, ввод и вывод из терминала, а второй пишет код, выполняющий основную логику программы.

Согласовав разделение, приступайте к написанию. Для написания вам необходимо локально создавать отдельную ветку, посвящённую вашему функционалу, и сливать с основной после завершения, после чего сливать с изменениями в удалённом репозитории и отправлять на него свои.

1. Определить какой-либо обособленный сегмент функциональности, который вы должны реализовать (например, проверку правильности входящих аргументов)
2. Создать ветку experimental и перейти в неё
3. Реализовать данный сегмент и провести фиксацию с \_развёрнутым\_ сообщением к коммиту
4. Слить ветку с локальным master
5. Запросить изменения, переданные вторым разработчиком, с удалённого репозитория
6. Провести проверку изменений, если таковые были, посмотреть и обсудить код со вторым разработчиком
7. Слить изменения из удалённого репозитория с локальным
8. Вернуться в подпункт а

Не забывайте, что от вас ожидается командная работа – вы должны, во-первых, писать код с целью того, чтобы второй разработчик его мог прочитать, и, во-вторых, помогать друг другу совместить ваши изменения.

1. **Требования к проекту**
   1. В рамках проекта вы можете использовать как и терминал, так и IDE для управления репозиторием, однако при сдаче вы должны знать и уметь пользоваться командами ‘git status’, ‘git add’, ‘git commit’, ‘git pull’, ‘git push’, ‘git branch’, ‘git checkout’, ‘git log’, ‘git merge’
   2. При выполнении коммитов описание следует выполнять согласно вышеприведённым рекомендациям.
   3. Итоговый проект должен находиться на удалённом репозитории под веткой с названием согласно вашим фамилиям
2. Варианты задания  
   Более сложные задания дают больше дополнительных баллов. Дублирование заданий в группе не разрешается, если не сказано иное в отдельных случаях.  
     
   1. Простой калькулятор  
        
      Реализовать простой скрипт, получающий на вход консольной строки два числа и оператор (+, -, \*, /, %, \*\*), выполняющий соответствующую операцию и возвращающий результат в консоль.
   2. Проверка числа на простоту  
        
      Реализовать программу, получающую на вход число (или предлагающей пользователю ввести интерактивно, если аргумента не было предоставлено), и возвращающая все простые делители этого числа.
   3. Счётчик слов  
        
      Реализовать программу, получающую на вход параграф текста (или предлагающей пользователю ввести интерактивно, если аргумента не было предоставлено), и возвращающая количество уникальных слов в тексте.
   4. Игра «Угадай число»  
        
      Программа загадывает случайное число от 1 до 100, и пользователь должен угадать его за определённое количество попыток. Количество попыток нужно получить из аргументов командной строки. При каждой попытке нужно сообщить пользователю сколько попыток осталось, а также то, больше ли загаданное число или меньше того, что предложил пользователь.
   5. Конвертер температур  
        
      Программа должна получать в качестве аргументов командной строки число и единицу измерения, Цельсий C, Фаренгейт F или Кельвин K. В терминал выводится конвертация в две остальные единицы измерения. Проверка на правильность значений обязательна.
   6. Счётчик слов+  
        
      Улучшенная версия счётчика слов также должна вывести частоту каждого слова в тексте, например:  
        
      [слово] : [количество] [частота] <\*пропорция\*>  
        
      «Мама мыла раму. Раму давно пора было помыть.»  
        
      раму : 2 0,25 \*\*\*  
      мама : 1 0,125 \*  
      давно : 1 0,125 \*  
      мыла : 1 0,125 \*  
      пора : 1 0,125 \*  
      было : 1 0,125 \*  
      помыть: 1 0,125 \*
   7. Генератор случайных паролей  
        
      Написать программу, генерирующую случайный пароль заданной длины, состоящий из заглавных и строчных букв, специальных символов и цифр. Длина задаётся аргументом командной строки
   8. Генератор фигур  
        
      На вход программы передаются размер фигуры (в строчках) и тип (треугольник, квадрат, трапеция, шестиугольник). Вывести в консоль текстовый рисунок искомой фигуры  
        
      Например:  
        
      Треугольник, 5  
        
       \*  
       \* \* \*  
       \* \* \* \* \*  
       \* \* \* \* \* \* \*  
      \* \* \* \* \* \* \* \* \*