



МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технологический университет «СТАНКИН»
(ФГБОУ ВО «МГТУ «СТАНКИН»)

Институт цифровых
интеллектуальных систем

Кафедра
компьютерных систем управления

Дисциплина «Основы системного программного обеспечения»

Отчет по лабораторной работе №1

Выполнила

студент гр. АДБ-21-08:

(дата)

(подпись)

Марковец Д.С.

Проверил

к.т.н., доцент

(дата)

(подпись)

Ковалев И.А.

Москва 2024 г.

Оглавление

Работа в системе контроля версий Git	4
Шаг 1. Создание учетной записи на github.com	5
Шаг 2. Изменение названия главной ветки	5
Шаг 3. Создание нового репозитория.....	6
Шаг 4. Установка git.....	7
Шаг 5. Создание локального репозитория	7
Шаг 6. Фиксация изменений в области заготовленных файлов.....	11
Шаг 6. Отправка коммитов на сервер	12
Шаг 7. Запрос изменений с сервера	13
Шаг 8. Перешлем локальный коммит на сервер	14
Шаг 9. Создание новой ветки	17
Шаг 10. Слияние веток	18
Шаг 11. Просмотр изменений и разрешение конфликтов	20
Шаг 12. Удаление веток на сервере	21
Шаг 13. Возврат к предыдущему состоянию	21
Шаг 14. Исправление коммита.....	22
Шаг 15. Отправка только нужных файлов на сервер.....	23
Вывод по лабораторной работе.....	27

Цель лабораторной работы: знакомство с системами контроля версий, изучение основ работы Git.

Задачи лабораторной работы:

1. Создание локального репозитория
2. Изучение основ Git и основных команд для работы с репозиторием

Работа в системе контроля версий Git

При выполнении лабораторной работы некоторые скриншоты отсутствуют. Аккаунты ArgonDragonConstellation(rex011101@gmail.com) и PurpureDragonConstellation(trimeresuruspurpure@gmail.com) принадлежат одному человеку. (При необходимости есть возможность с помощью команды `git clone` клонировать созданный репозиторий и продолжить работу с ним уже на другой машине.) `Git status` позволяет нам находить новые не отслеживаемые файлы или отслеживаемые файлы с изменениями в них. Команда `cd` позволяет нам перейти в созданный репозиторий, `git log` позволяет отслеживать все изменения сделанные в репозитории (слияние, удаление, создание веток, созданные коммиты и т.д.).

Шаг 1. Создание учетной записи на github.com

Так как мы уже работали с github учётная запись имеется.(2 учётных записи)



ArgonDragonConstellation (ArgonDragonConstellation)

Your personal account



PurpureDragonConstellation

Дарья

Рис.1.Учетная запись

Шаг 2. Изменение названия главной ветки

В settings уже было изменено название главной ветки с "main" на "master".

Repository default branch

Choose the default branch for your new personal repositories. You might want to change the default name due to different workflows, or because your integrations still require "master" as the default branch name. You can always change the default branch name on individual repositories. [Learn more about default branches.](#)

Рис.2.Название главной ветки master


Шаг 3. Создание нового репозитория

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *

 PurpureDragonConstellation

Repository name *

OSPOLabs

✓ OSPOLabs is available.

Description (optional)

Labs to study OSPO program

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Рис.3.Новый репозиторий.Создание

Для нового репозитория задаём имя, можно добавить описание. Тип репозитория, не смотря на наличие закрытого, для лабораторных работ выбираем открытый "public". Отмечаем добавление в репозиторий файла README, чтобы тому, кто заходит в репозиторий первым делом открывалось описание проекта.

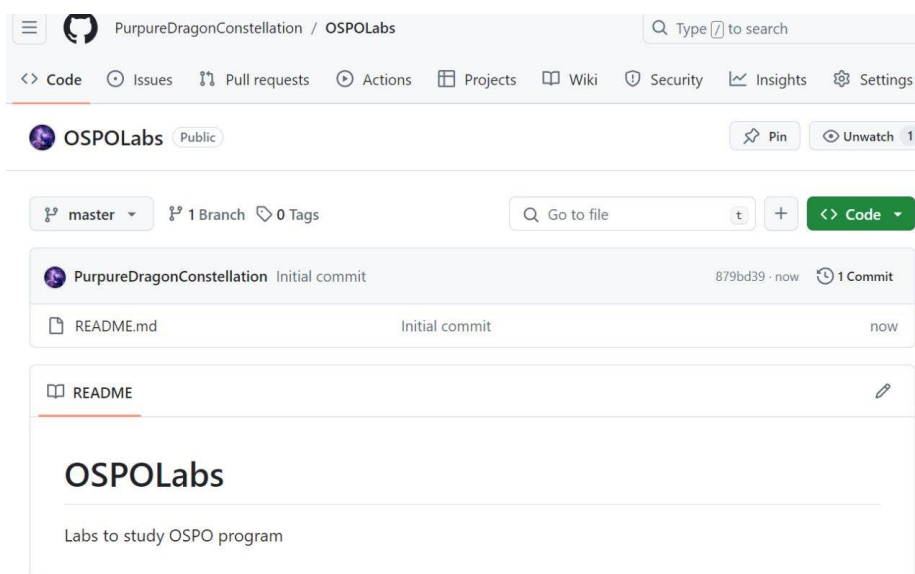


Рис.4.Новый репозиторий

Шаг 4. Установка git

git установлен (на персональный ПК), так что можно приступать к следующему шагу лабораторной работы.

Если не установлен git, нужно скачать и установить необходимую версию <https://git-scm.com/downloads>. Проверить это просто. Открыв командную строку и написав git, посмотреть: если появилась куча команд, то git есть. Если появилась ошибка, то git нужно установить.

Удаление предыдущей записи git на ПК

Если работа проходит не на своем ПК и там уже был залогированный Git, то лучше удалить предыдущую запись. Проще это сделать через локальные записи, для этого необходимо нажать Win+R и вписать в появившемся окошке «Выполнить» команду control.

Далее надо начать печатать в строке поиска «Учетные записи» и выбрать «Учетные записи пользователей»

Выбрать «Администрирование учетных записей»

Далее выбрать «Учетные записи Windows» и в «Общие учетные записи» найти поле с github, развернуть его и нажать удалить. Можно создать несколько таких записей, но лучше ограничиться одной.

Шаг 5. Создание локального репозитория

Создадим теперь локальную папку, которая будет связана с нашим облачным git хранилищем. В общем случае облачный git может представлять также локальный сервер, объединенный в единую информационную сеть со всеми ПК.

Создадим на диске папку со своей фамилией, лучше в «Документы» или «Загрузка», так будет меньше конфликтов с правом доступа. Создадим в ней файл test.txt и напомним в нем свою фамилию. *Примечание: названия лучше

писать латиницей, с кириллицей тоже будет работать, но отображаться будет непонятно.

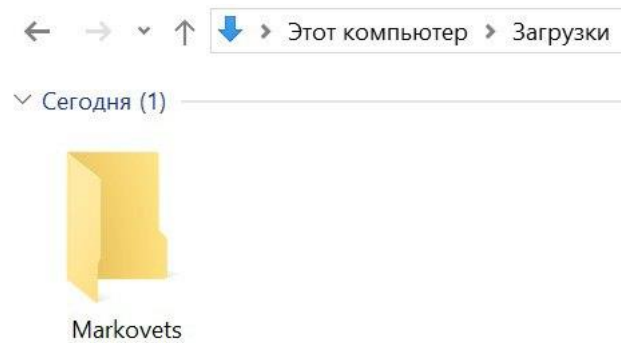


Рис.5.Локальная папка

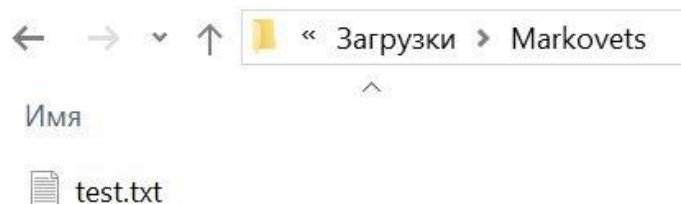


Рис.6.Файл test.txt

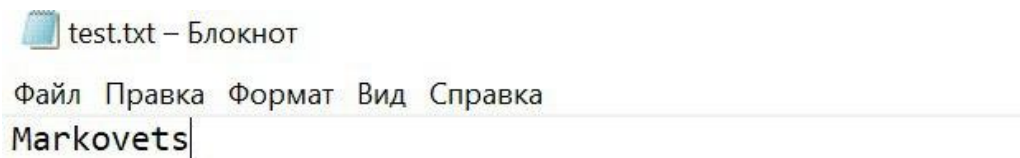
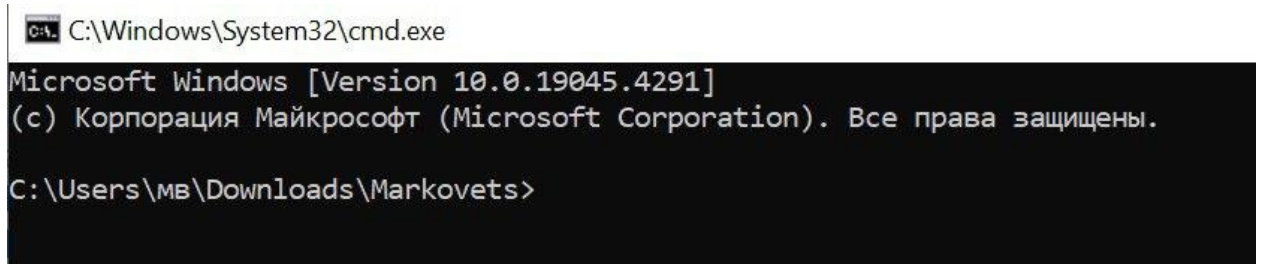


Рис.7.Содержимое файла test.txt

Откроем командную строку «cmd» и перейдем с помощью команды «cd» в созданный в каталог. Проще всего это сделать, выделив путь в проводнике, написать вместо него cmd и нажать enter.



Рис.8.Замена пути cmd



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.4291]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.
C:\Users\mb\Downloads\Markovets>
```

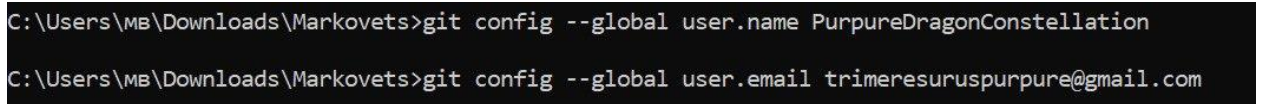
Рис.9.Открытие командной строки

*Примечание: при работе с git можно использовать PowerShell, GitBash, Терминал и даже различные IDE, но мы будем работать с обычной командной строкой.

Настроим самые важные опции и параметры: имя пользователя и адрес электронной почты (лучше указать почту, на которую был зарегистрирован gitHub). **Запустим команды (со своими данными), например:**

git config --global user.name PurpureDragonConstellation

git config --global user.email trimeresuruspurpure@gmail.com



```
C:\Users\mb\Downloads\Markovets>git config --global user.name PurpureDragonConstellation
C:\Users\mb\Downloads\Markovets>git config --global user.email trimeresuruspurpure@gmail.com
```

Рис.10.Настройка привязки

ОБЯЗАТЕЛЬНО надо выполнить эти команды, иначе все изменения могут помечаться почтой и именем того, кто раньше представился.

Перейдем в созданную папку в командной строке, используя команду cd.

Теперь необходимо проинициализировать эту папку как git репозиторий

Введем в командной строке:

git init

Командная строка вернет сообщение, что **проинициализирована пустая git директория**.

```
C:\Users\mb\Downloads\Markovets>git init
Initialized empty Git repository in C:/Users/mb/Downloads/Markovets/.git/
```

Рис.11.Инициализация пустой git директории

Теперь используя проводник windows создадим в своей папке любой текстовый файл и пропишем в него имя.

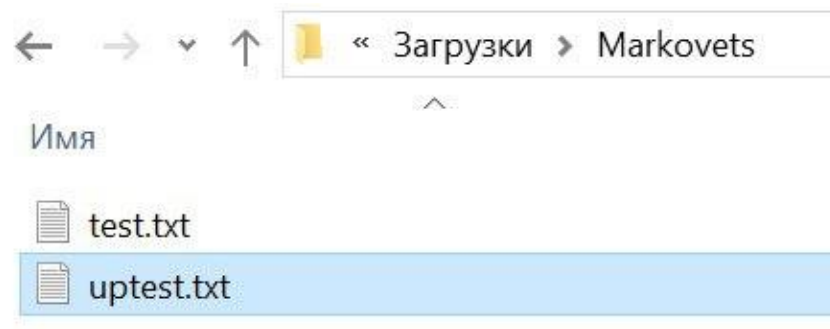


Рис.12.Новый текстовый файл

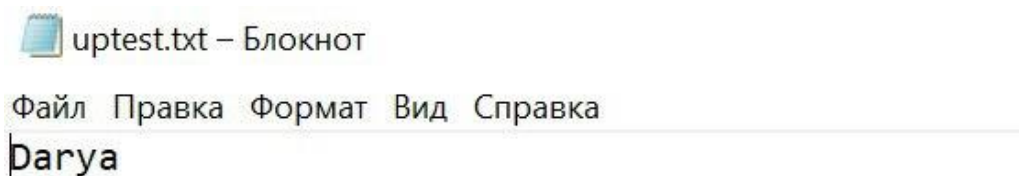


Рис.13.Имя в файле

Введите команду:

git status

и вам отобразиться сообщение, что есть новый файл, но он не отслеживается. Это значит, что файл новый и система еще не знает, нужно ли следить за изменениями в файле или его можно просто игнорировать. Для того, чтобы начать отслеживать новый файл, нужно его специальным образом объявить.

```

C:\Users\MB\Downloads\Markovets>git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test.txt
    uptest.txt

nothing added to commit but untracked files present (use "git add" to track)

```

Рис.14. Неотслеживаемые новые файлы

В git есть концепция области подготовленных файлов. Можно представить ее как холст, на который наносят изменения, которые нужны в коммите. Сперва он пустой, но затем мы добавляем на него файлы (или части файлов, или даже одиночные строчки) командой `add` и, наконец, коммитим все нужное в репозиторий (создаем слепок нужного нам состояния) командой `commit`.

Шаг 6. Фиксация изменений в области заготовленных файлов

Если хотим добавить только **один файл**, можно написать:

`git add test.txt`

Если **измененных файлов много**, то можно сделать так:

`git add .`

Выполнив нужную команду, снова посмотрим статус репозитория (`git status`).

Используем `git add .`

```

C:\Users\MB\Downloads\Markovets>git add .

C:\Users\MB\Downloads\Markovets>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   test.txt
    new file:   uptest.txt

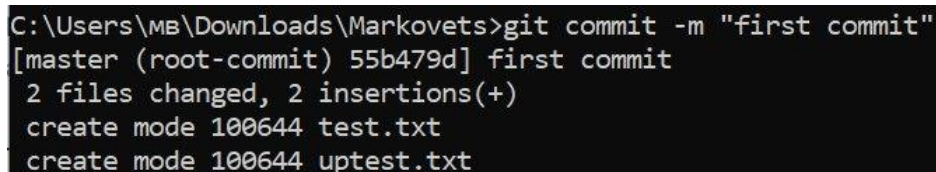
```

Рис.15. Добавление файлов

Файл готов к **коммиту**. Сообщение о состоянии также говорит нам о том, какие изменения относительно файла были проведены в области подготовки — в данном случае это новый файл, но файлы могут быть модифицированы или удалены.

Теперь **закоммитим** (отправка правок в репозиторий) **наши файлы**. Коммит представляет собой состояние репозитория в определенный момент времени как некий опечаток во времени. Коммит отмечается hash суммой и к которому мы можем в любой момент времени вернуться. Чтобы зафиксировать изменения, нам нужно хотя бы одно изменение в области подготовки (мы как раз создали новый файл):

`git commit -m "first commit"`



```
C:\Users\mb\Downloads\Markovets>git commit -m "first commit"
[master (root-commit) 55b479d] first commit
 2 files changed, 2 insertions(+)
 create mode 100644 test.txt
 create mode 100644 uptest.txt
```

Рис.16.Коммит

Команда создаст новый коммит со всеми изменениями из области подготовки. Считается хорошей практикой делать коммиты часто и всегда писать содержательные комментарии. Вызовем `git status`.

Шаг 6. Отправка коммитов на сервер

Перед тем как перейти к более продвинутым вещам, рассмотрим, как **связать наш локальный репозиторий с облачным репозиторием**, который мы создали на github.com

Чтобы связать наш локальный репозиторий с репозиторием на GitHub, выполним следующую команду в терминале.

`git remote add origin`

<https://github.com/PurpleDragonConstellation/OSPOLabs.git>

Проект может иметь несколько удаленных репозиторий одновременно. Чтобы их различать, мы дадим им разные имена. Обычно **главный репозиторий называется origin**.

```
C:\Users\mb\Downloads\Markovets>git remote add origin https://github.com/PurpleDragonConstellation/OSPOLabs.git
```

Рис.17.Связывание репозиторий: локального и GitHub

Можно вызвать команду, для просмотра, к какому проекту мы **подключены**

`git remote -v`

```
C:\Users\mb\Downloads\Markovets>git remote -v
origin https://github.com/PurpleDragonConstellation/OSPOLabs.git (fetch)
origin https://github.com/PurpleDragonConstellation/OSPOLabs.git (push)
```

Рис.18.Просмотр к какому проекту мы подключены

Если вдруг на этом месте показывается несколько репозиторий, то скорее всего верхние команды привели к запутыванию веток, это можно разрешить следующей командой. Также эта команда может пригодиться, если возникли ошибки, связанные с unrelated-histories.

`git pull origin master --allow-unrelated-histories`

Занустите `git remote -v` снова

Если репозиторий указан неверно, его можно удалить командой (но только если он указан неверно)

`git remote rm origin`

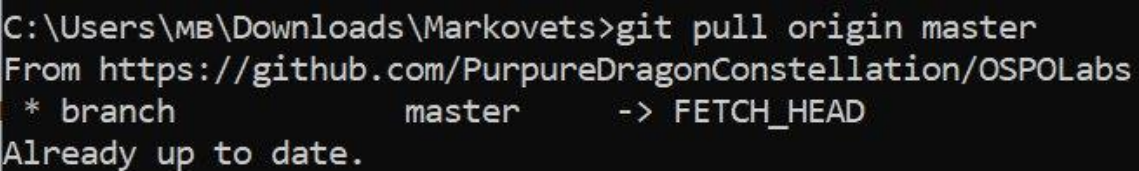
После необходимо снова подключить верный репозиторий.

Шаг 7. Запрос изменений с сервера

В нашем случае у нас есть файл и на сервере и локальном репозитории, поэтому нам вначале надо получить все данные с нашего репозитория, а потом уже их заливать на сервер нашими локальными данными.

Мы сделали изменения в нашем репозитории (создав файл `readme`), следовательно **другие пользователи могут скачать изменения при помощи команды `pull`**.

`git pull origin master`



```
C:\Users\mb\Downloads\Markovets>git pull origin master
From https://github.com/PurpureDragonConstellation/OSPOLabs
 * branch      master      -> FETCH_HEAD
Already up to date.
```

Рис.19. Извлечение главной ветки из удалённого вызываемого источника в нашу текущую ветку.

Это влияет только на нашу текущую ветку, а не на нашу локальную главную ветку. ***Git pull*** — это, по сути, комбинация ***git fetch*** (команда в распределённой системе контроля версий Git, которая используется для скачивания изменений из удалённого или локального репозитория в свой локальный репозиторий) и ***git merge*** (выполняет слияние отдельных направлений разработки, созданных с помощью команды `git branch`, в единую ветку). Она извлекает удалённую ветку, а затем объединяет её с нашей текущей веткой.

Шаг 8. Перешлем локальный коммит на сервер

Теперь **отправим коммит на сервер**, команда, предназначенная для этого — `push`. Она принимает два параметра: имя удалённого репозитория (мы назвали наш `origin`) и ветку, в которую необходимо внести изменения (`master` — это ветка по умолчанию для всех репозиториях).

`git push origin master`

Появилось окно, где нужно авторизоваться на `github.com`

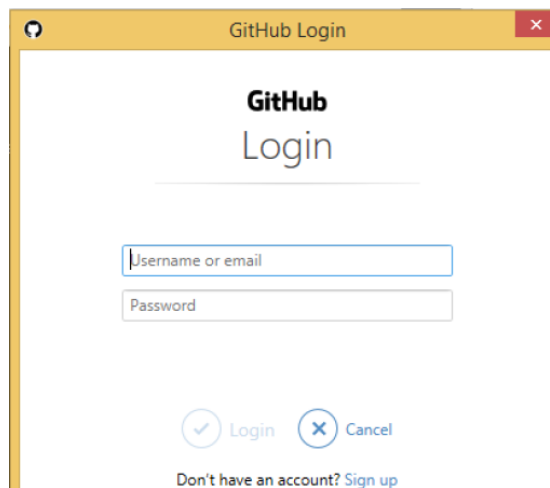


Рис.20.Окно авторизации в GitHub

Если появляется ошибка, связанная с `unrelated-histories`, то нужно воспользоваться командой, которая была описана выше.

Окно может выглядеть и по-другому, где предлагается ввести персональный токен или зайти через браузер. Выбираем вариант использования браузера – надо нажать `Sign in with your browser`



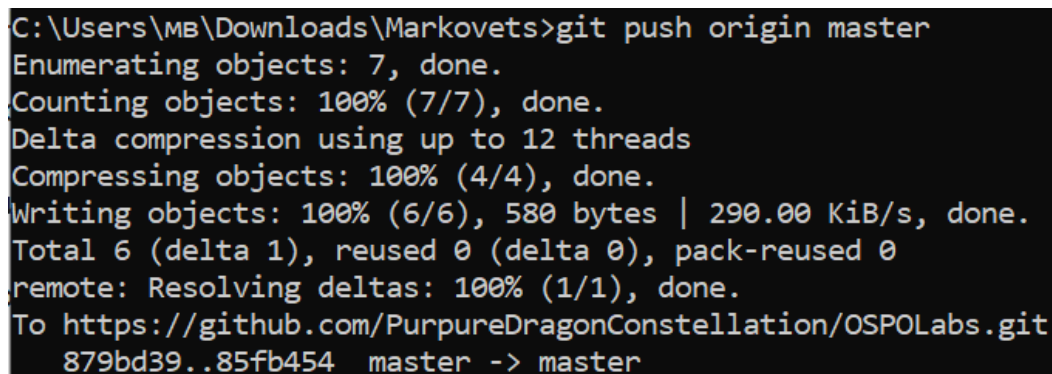
Рис.21.Другое окно авторизации в GitHub

Откроется страница в браузере по умолчанию (если был использован для регистрации этот же браузер и не было выхода из `github.com`). (Как раз этот случай.)

Если это другой браузер, то надо залогиниться на github под своим аккаунтом)

На странице надо нажать «Authorize GitCredentialManager»

Все правильно, изменения отправляются на сервер.



```
C:\Users\мв\Downloads\Markovets>git push origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 580 bytes | 290.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/PurpleDragonConstellation/OSPOLabs.git
  879bd39..85fb454  master -> master
```

Рис.22. Изменения отправлены на сервер

Если получаем ошибку 403, скорее всего git не понимает адрес удаленного репозитория, либо неверно введены логин/пароль, можно попробовать перелогиниться, вызвав команды

`git config --global credential.github.com.interactive always`

теперь снова пытаемся сделать push и вводим новые логин/пароль

`git config --global credential.github.com.interactive auto`

Наш комит будет иметь hash 85fb454057575bc8ad04a13e45be4ba1acf826b1, по которому можно будет вернуться к нему в случае необходимости

Посмотрим наши изменения с использованием команды

`git log`


```

C:\Users\MB\Downloads\Markovets>git log
commit 85fb454057575bc8ad04a13e45be4ba1acf826b1 (HEAD -> master, origin/master)
Merge: 55b479d 879bd39
Author: PurpureDragonConstellation <trimeresuruspurpure@gmail.com>
Date:   Fri May 31 04:40:04 2024 +0300

    Merge branch 'master' of https://github.com/PurpleDragonConstellation/OSPOLabs

commit 55b479dd1eee085db67339275096d2c7efff26cf
Author: PurpureDragonConstellation <trimeresuruspurpure@gmail.com>
Date:   Fri May 31 04:11:01 2024 +0300

    first commit

commit 879bd394eebda066d7efb84e9214f3921f91ef1c
Author: Дарья <133849184+PurpureDragonConstellation@users.noreply.github.com>
Date:   Fri May 24 03:22:53 2024 +0300

    Initial commit

```

Рис.23. Просмотр изменений с помощью git log

Видны коммиты по изменениям в файле Readme.md и в файле с именем (first commit), чтобы выйти из просмотра лога надо нажать q.

Можно посмотреть изменения на github.com

Если нажать на «commit», то можно увидеть все наши изменения.

Шаг 9. Создание новой ветки

На этом моменте скриншоты по работе с командной строкой не сохранились.

Но продолжим описание работы, ведь это было сделано.

Во время разработки новой функциональности считается хорошей практикой работать с копией оригинального проекта, которую называют веткой. Ветви имеют свою собственную историю и изолированные друг от друга изменения

до тех пор, пока мы не решаем слить изменения вместе. Это происходит по набору причин:

- *Уже рабочая, стабильная версия кода сохраняется.*
- *Различные новые функции могут разрабатываться параллельно разными программистами.*

- Разработчики могут работать с собственными ветками без риска, что кодовая база поменяется из-за чужих изменений.
- В случае сомнений, различные реализации одной и той же идеи могут быть разработаны в разных ветках и затем сравниваться.

Обычно в проекте создается минимум 2 ветки (branch) – master и ветка для разработки. Ветка master создается как само собой разумеющееся, а дополнительные ветки создаются по желанию пользователя (локально), заливаются они в удаленный репозиторий уже с учетом доступных прав.

1)Создадим новую ветку second:

git branch second

2)Можно посмотреть в какой ветке находимся, используя команду

git branch

3)Переключимся на другую ветку, используя команду

git checkout second

4)Создадим новый файл в нашем локальном репозитории и напомним в нем свою фамилию, добавим в область подготовленных файлов, закоммитим и отправим на сервер, только стоит учитывать, что у нас теперь ветка не master, а second.

5)Зайдем на github и посмотрим, что в ветке master нет нашего файла, а в ветке second он присутствует.

Шаг 10. Слияние веток

Переключимся снова на ветку master

Если через проводник windows зайти в папку с проектом, то файла test_new.txt не будет видно, потому что мы переключились обратно на ветку master, в которой такого файла не существует. Чтобы он появился, нужно

воспользоваться merge для **объединения веток** (применения изменений из ветки second к основной версии проекта).

git merge second

Если все прошло без ошибок, можно **удалить ветку**.

git branch -d second

Предположим, что в двух ветках могут быть одинаковые файлы и над ними работают разные разработчики

Создадим ветку с названием newdev

git branch newdev

Переключимся на неё

git checkout newdev

Добавим в файл с именем отчество

Зафиксируем изменения

git pull origin newdev

Закоммитим изменения

git push origin newdev

Попробуем переключиться обратно на ветку master и посмотреть файл: в нем только имя, отчество в другой ветке

Объединим ветки

git merge newdev

Откроем файл снова, там есть и имя, и отчество

Переключимся на ветку newdev и удалим в отчестве несколько букв

Зафиксируем, закоммитим

Переключимся на ветку **master**, добавим к отчеству несколько букв

Зафиксируем, закоммитим

Объединим ветки

git merge newdev

Теперь ничего не получится, т.к. есть изменения в обеих ветках

Шаг 11. Просмотр изменений и разрешение конфликтов

Нужно разрешить конфликты.

Наберем команду, для просмотра изменений

git diff

В моём случае конфликт исправился автоматически.

Иногда конфликты исправляются автоматически, но обычно с этим приходится разбираться вручную — решать, какой код остается, а какой нужно удалить.

*Система не смогла разрешить конфликт автоматически, значит, это придется сделать разработчикам. Приложение отметило **строки, содержащие конфликт**, например:*

<<<<<< HEAD

Sergeevnaaa

=====

Serge

>>>>>> newdev

Над разделителем ===== мы видим последний (HEAD) коммит, а под ним — конфликтующий. Таким образом, мы можем увидеть, чем они отличаются и решать, какая версия лучше. Или вовсе написать новую. В этой ситуации

мы так и поступим, перепишем все, удалив разделители (HEAD, ==, <<<, >>>), и дадим git понять, что закончили.

Процесс может быть довольно утомительным и может быть очень сложным в больших проектах. Многие разработчики предпочитают использовать для разрешения конфликтов клиенты с графическим интерфейсом.

Выбираем нужную строку (какую строку оставить, может вообще все хотим оставить), удаляем разделители, сохраняем файл, фиксируем изменения, коммитим, отправляем на сервер.

Удалим ветку newdev

Шаг 12. Удаление веток на сервере

Если у нас есть несмерженная ветка, но мы хотим ее удалить, необходимо использовать команду:

`git branch -D second`

Если посмотреть на github, то наши ветки там остались.

Удалим ветки из github

`git push origin --delete newdev`

`git push origin --delete second`

Все ветки удалены.

Шаг 13. Возврат к предыдущему состоянию

Гит позволяет вернуть выбранный файл к состоянию на момент определенного коммита. Это делается уже знакомой нам командой `checkout`, которую мы ранее использовали для переключения между ветками. Но она также может быть использована для переключения между коммитами.

Чтобы **посмотреть все комиты**, можно использовать команду

git log

Или можно использовать github или любой клиент.

Выберем любой **коммит, на который хотим откатиться**, достаточно указать его первые несколько символов (перед выполнением посмотрите свои файлы, чтобы запомнить, что там находится)

git checkout 2690e2987bdd5a886c7307c4f9a3f4bbcd0cd13f

Если посмотреть файлы теперь, то увидим, что они поменялись – мы вернулись назад.

Чтобы это произошло создастся псевдо-ветка начинающаяся на этом коммите, посмотрим ветки.

Если работаем с кем-то над одним проектом, то правильнее переключиться на работающий коммит, проверить что все в нем работаете, создать от него новую ветку и провести слияние. Либо просто перенести изменения в файлы, которые сейчас лежат в HEAD.

Все команды, которые позволяют поменять HEAD git, при этом удаляя ненужные коммиты, могут являться опасными, в плане потери времени из-за путаницы изменений файлов.

Откатимся обратно на master ветку

Шаг 14. Исправление коммита

Возможно ситуация, что мы **закрепили файлы** (поменяйте что-то в файле и закрепите их, вызовите git status), **но еще не коммитили, хотим убрать файлы из области закрепления**, вызовем команду:

git reset HEAD

Файлы останутся такими же, но уйдут из области закрепления (вызовем git status) и снова будет показано, что есть измененные файлы. Теперь снова добавим в область закрепления и закоммитим, но еще не отправляем на сервер,

можно вызвать ту же команду и тогда все, что осталось незакоммиченным будет удалено.

Если хотим **изменить комментарий** можно вызвать команду:

git commit --amend

Откроется консоль в редакторе Vi, чтобы **начать что-то менять**, надо нажать клавишу Insert, **когда изменения в комментарий внесены**, необходимо снова нажать Insert. Чтобы **сохранить изменения**, нажмем Esc, потом **нажмем двоекочие** (появится внизу экрана), **напишем w!** и **нажмем Enter**. Для **выхода** снова нажмем двоеочие, введем q! и нажмем Enter.

Шаг 15. Отправка только нужных файлов на сервер

В большинстве проектов есть файлы или целые директории, в которые мы не хотим (и, скорее всего, не захотим) коммитить. Мы можем удостовериться, что они случайно не попадут в git add -А при помощи файла «.gitignore»

Создаем вручную файл под названием «.gitignore» и сохраняем его в директорию проекта.

Внутри файла перечисляем названия файлов/папок, которые нужно игнорировать, каждый с новой строки.

Файл «.gitignore» должен быть добавлен, закоммичен и отправлен на сервер, как любой другой файл в проекте.

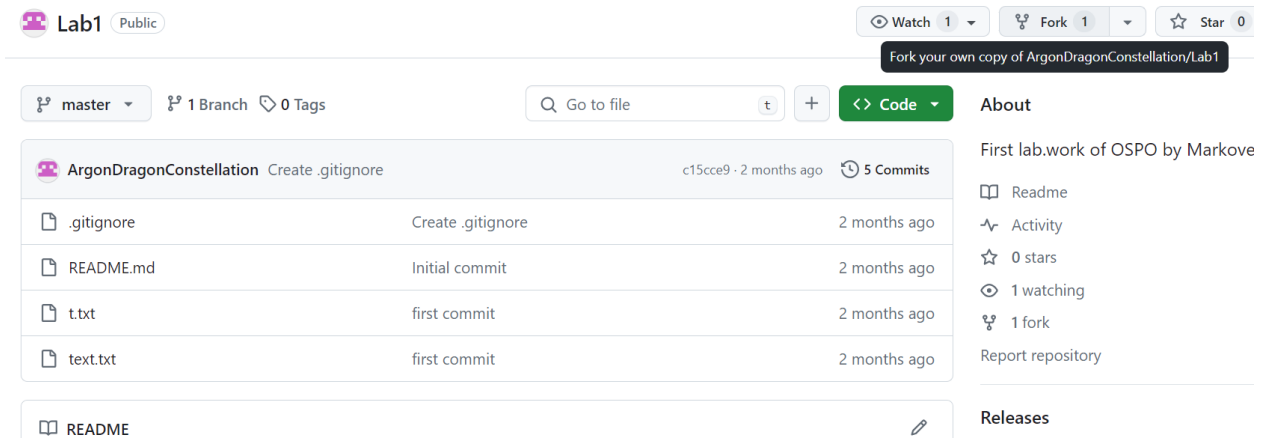
Впишем в «.gitignore» файл, который не хотим отправлять на сервер.

Зафиксируем изменения, закоммитим, отправим на сервер.

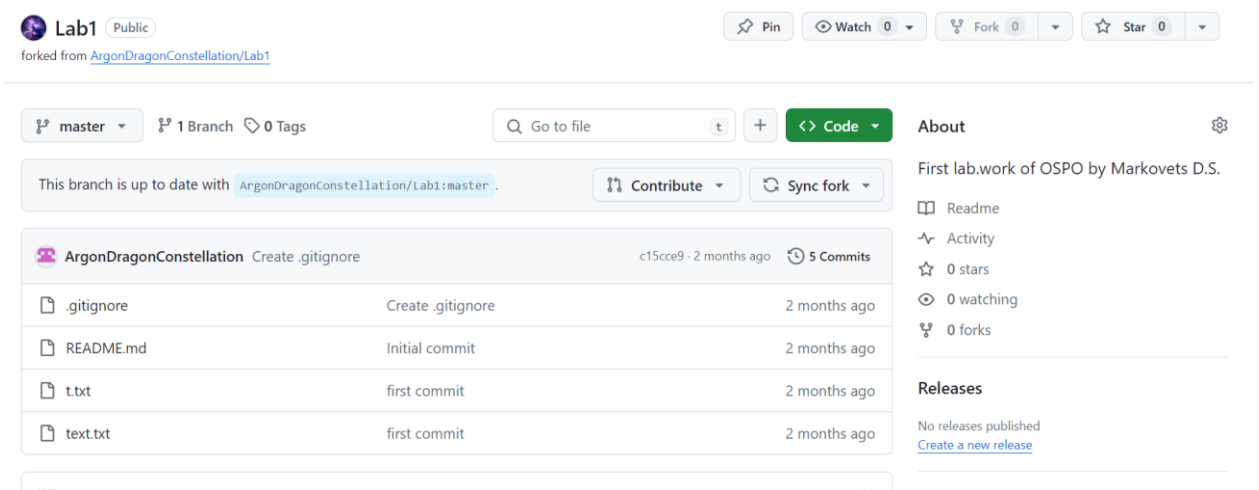
! Все изменения можно проводить непосредственно на github, там также можно делать коммиты, только фиксация изменений происходит при сохранении файла.

Выберем себе напарника, скинем друг другу адреса своих репозиторий (в моём случае напарник – второй аккаунт).

Переходим по полученным ссылкам на репозиторий и нажимаем fork.



Теперь этот репозиторий добавляется к нам в проекты github



Далее через консоль клонируем этот репозиторий на локальный компьютер, предварительно создадим папку, в которую будем клонировать и в консоли перейдём в нее

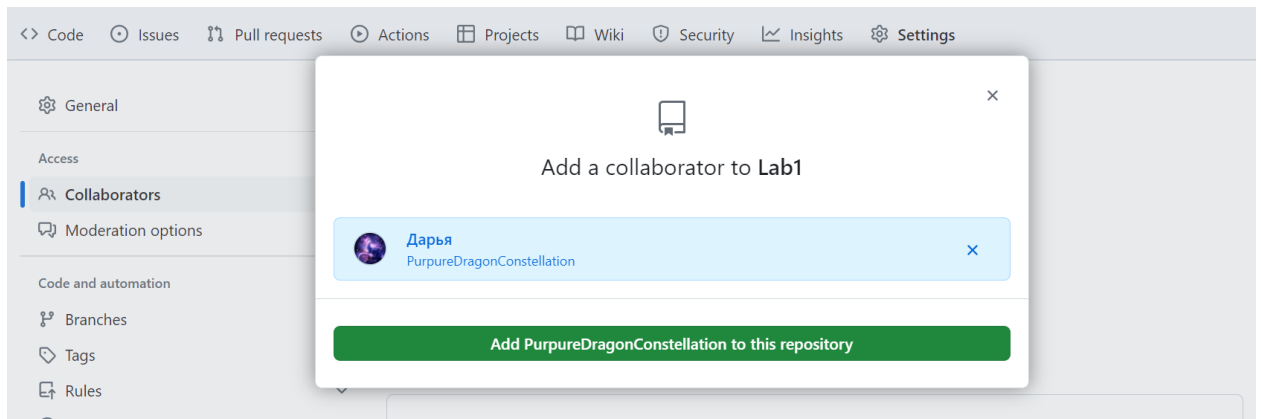
[git clone https://github.com/PurpureDragonConstellation/Lab1.git](https://github.com/PurpureDragonConstellation/Lab1.git)

Таким образом мы создали скопировали проект и работаем с ним, но запись в проект напарника не происходит

Необходимо дать права доступа для записи в наш репозиторий

Для этого перейдем на github в раздел Settings, раздел Collaboration

Найдем в строке поиска того, кому хотим дать доступ в проект



Создадим новую папку и перейдем в нее в консоли

Сделаем clone проекта, к которому нам дали доступ

git clone «адрес репозитория»

[git clone https://github.com/ArgonDragonConstellation/Lab1.git](https://github.com/ArgonDragonConstellation/Lab1.git)

Попробуем теперь что-то поменять, закоммитемся и отправим на сервер.

Поработаем над одним файлом

Добавим новый файл, отправим его на сервер, чтобы, когда все разработчики сделают pull себе он появился локально у них

Сделаем push нескольких новых файлов, а также изменений добавленных файлов одним из разработчиков

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.2486]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

F:\PurpureDragonConstellationfork>git clone https://github.com/PurpureDragonConstellation/Lab1
Cloning into 'Lab1'...
remote: Enumerating objects: 39, done.
remote: Counting objects: 100% (39/39), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 39 (delta 11), reused 34 (delta 9), pack-reused 0
Receiving objects: 100% (39/39), done.
Resolving deltas: 100% (11/11), done.
```

```
Выбрать C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.2486]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

F:\PurpureDragonConstellationcollab>git clone https://github.com/ArgonDragonConstellation/Lab1
Cloning into 'Lab1'...
remote: Enumerating objects: 39, done.
remote: Counting objects: 100% (39/39), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 39 (delta 11), reused 34 (delta 9), pack-reused 0
Receiving objects: 100% (39/39), done.
Resolving deltas: 100% (11/11), done.
```

```
F:\PurpureDragonConstellation\Lab1>git branch darya
F:\PurpureDragonConstellation\Lab1>git checkout darya
Switched to branch darya

F:\PurpureDragonConstellation\Lab1>git status
On branch darya
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test.txt
```

```
F:\PurpureDragonConstellation\Lab1>git commit -m "darya first"
[PurpureDragonConstellation caddbfa] darya first
1 file changed, 1 insertion(+), 1 deletion(-)
```

Итог:

ArgonDragonConstellation / Lab1

Type ↵ to search

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

Lab1

Public

Unwatch 1

darya ▾

2 Branches

0 Tags

Go to file

t

+

<> Code ▾

This branch is 2 commits ahead of master .

Contribute ▾

PurpureDragonConstellation

Update new.txt

71597d2 · 33 minutes ago

7 Commits

.gitignore	Create .gitignore	2 months ago
README.md	Initial commit	2 months ago
new.txt	Update new.txt	33 minutes ago
t.txt	first commit	2 months ago
text.txt	first commit	2 months ago

Ссылка на репозиторий GitHub:

<https://github.com/ArgonDragonConstellation/Lab1.git>

Вывод по лабораторной работе

В результате данной работы мы познакомились с системами контроля версий, изучили основы работы с Git и основные команды Git. А также поработали в команде в репозитории.

Список использованной литературы:

1. Электронные издания научно-технической библиотеки, размещенные в разделе университета в ЭБС «Университетская библиотека онлайн»
2. Электронная библиотека научных публикаций «Российский индекс научного цитирования» // Режим доступа URL: elibrary.ru/
3. ЭБС «IPRBOOKS» группа компаний IPR MEDIA // Режим доступа URL: www.iprbookshop.ru
4. Интернет-ресурс кафедры «Компьютерные системы управления» МГТУ «СТАНКИН» - <http://www.ncsystems.ru/>
5. Методическое пособие: Лабораторная работа №1. Работа Git Hub // Режим доступа URL: <http://bit.ly/3c5IF5c>