Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
дисциплины «Программирование на Python»

Выполнил:
Середа Кирилл Витальевич
1 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Основы ветвления Git

Цель: исследование базовых возможностей по работе с локальными и удаленными ветками Git.

Краткое теоретическое обоснование

Ветка в Git — это простой перемещаемый указатель на один из таких коммитов. По умолчанию, имя основной ветки в Git — master . Как только вы начнёте создавать коммиты, ветка master будет всегда указывать на последний коммит. Каждый раз при создании коммита указатель ветки master будет передвигаться на следующий коммит автоматически.

HEAD – это указатель на коммит в вашем репозитории, который станет родителем следующего коммита. Для того, чтобы лучше понять это, рассмотрим пример репозитория, в котором сделано сделано шесть коммитов. HEAD указывает на коммит, относительного которого будет создана рабочая копия во время операции checkout . Другими словами, когда вы переключаетесь с ветки на ветку, используя операцию checkout , то в вашем репозитории указатель HEAD будет переключаться между последними коммитами выбираемых вами ветвей.

Создание веток выполняется с помощью команды git banch

Для переключения на существующую ветку выполняется команда git checkout

Удаленная ветка - это ветка, которая существует в удаленном репозитории и отслеживает состояние истории изменений в этом удаленном репозитории. Она может быть доступна для скачивания и обновления изменений между вашим локальным репозиторием и удаленным репозиторием. Удаленные ветки используются для совместной работы и синхронизации изменений между разными разработчиками и репозиториями.

Ветка отслеживания - это локальная ветка в Git, которая непосредственно связана с удаленной веткой. Ветка отслеживания автоматически отслеживает изменения в удаленной ветке и позволяет синхронизировать локальные изменения с удаленным репозиторием.

Для создания ветки отслеживания в Git, вы можете использовать команды git checkout или git switch с флагом -t (или --track).

Для слияния веток используется команда git merge. Для решения конфликтов при слиянии используется как ручной метод так и различные утилиты.

Ход выполнения:

1) Изучил теоретический материал

2) Создал общедоступный репозиторий и клонировал его на компьютер

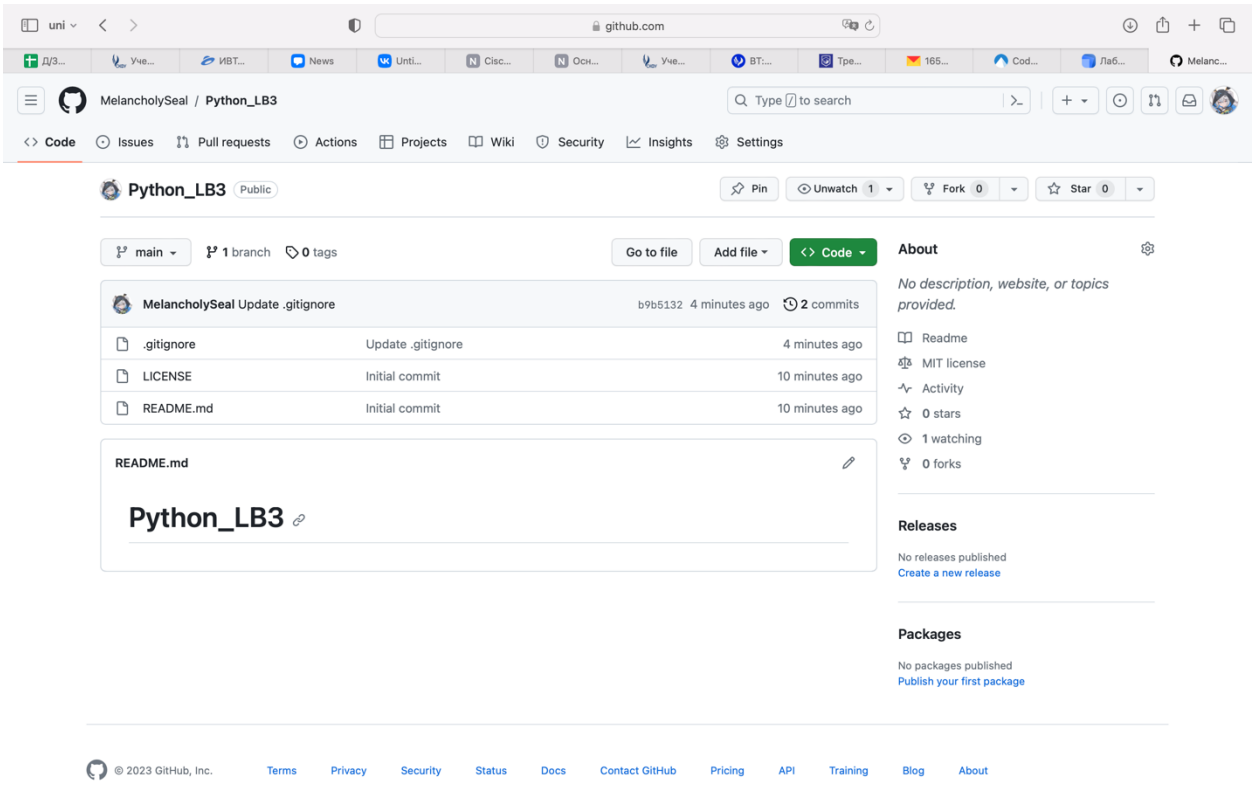

Рисунок 1 – Репозиторий на удаленном сервере

3) Проиндексировал первый файл и сделал коммит с комментарием "add 1.txt file".

```
nothing added to commit but untracked files present (use "git add" to track)
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git add 1.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git commit -m "add 1.txt file"
[main b3080f3] add 1.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 2 – Окно терминала после выполненных действий

4) Проиндексировал второй и третий файлы.

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git add 2.txt 3.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 3 - Проиндексировать второй и третий файлы.

5) Перезаписал последний коммит с новым комментарием "add 2.txt and 3.txt"

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git commit --amend -m "add 2.txt and 3.txt"
[main 0a110a7] add 2.txt and 3.txt
 Date: Tue Sep 26 16:17:28 2023 +0300
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1.txt
 create mode 100644 2.txt
 create mode 100644 3.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 4 - Окно терминала после выполненных действий

6) Создал новую ветку my_first_branch.

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git branch my_first_branch
```

Рисунок 5 - Окно терминала после выполненных действий

7) Перешел на ветку и создал новый файл in_branch.txt, закоммитил изменения.

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git switch my_first_branch
Switched to branch 'my_first_branch'
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % touch in_branch.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git add in_branch.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git commit -m "Add in_branch.txt"
[my_first_branch bc635d2] Add in_branch.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 6 - Окно терминала после выполненных действий

8) Вернулся на основную ветку

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git switch main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 7 - Окно терминала после выполненных действий

9) Создал и перешел на новую ветку new_branch

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git checkout -b new_branch
Switched to a new branch 'new_branch'
```

Рисунок 8 - Окно терминала после выполненных действий

10) Внес изменения в файл 1.txt, добавив строчку "new row in the 1.txt file", и закоммитил изменения

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % echo "new row in the 1.txt file" >> 1.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git add 1.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git commit -m "Add a new row to 1.txt"
[new_branch f15f5e8] Add a new row to 1.txt
 1 file changed, 1 insertion(+)
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 9 - Окно терминала после выполненных действий

11) Перешел на ветку main и выполнил слияние веток master и my_first_branch, а затем слияние веток main и new_branch

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git merge -m"second merge" new_branch
Merge made by the 'ort' strategy.
```

Рисунок 10 - Окно терминала после выполненных действий

12) Удалил ветки my_first_branch и new_branch

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git branch -d my_first_branch
Deleted branch my_first_branch (was bc635d2).
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git branch -d new_branch
Deleted branch new_branch (was f15f5e8).
```

Рисунок 11 - Окно терминала после выполненных действий

13) Создал ветки branch_1 и branch_2

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git checkout -b branch_1
Switched to a new branch 'branch_1'
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git checkout -b branch_2
Switched to a new branch 'branch_2'
```

Рисунок 12 - Окно терминала после выполненных действий

14) Перешел на ветку branch_1 и внес изменения в файлы 1.txt и 3.txt, как описано, затем закоммитил изменения

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git switch branch_1
Switched to branch 'branch_1'
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % echo "fix in the 1.txt" > 1.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % echo "fix in the 3.txt" > 3.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git add 1.txt 3.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git commit -m "Fixes in branch_1"
[branch_1 7ab84cb] Fixes in branch_1
 2 files changed, 2 insertions(+), 1 deletion(-)
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 13 - Окно терминала после выполненных действий

15) Перешел на ветку branch_2 и внес изменения в файлы 1.txt и 3.txt, как описано, затем закоммитил изменения

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git switch branch_2
Switched to branch 'branch_2'
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % echo "My fix in the 1.txt" > 1.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % echo "My fix in the 3.txt" > 3.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git add 1.txt 3.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git commit -m "My Fixes in branch_2"
[branch_2 39c772d] My Fixes in branch_2
 2 files changed, 2 insertions(+), 1 deletion(-)
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 14 - Окно терминала после выполненных действий

16) Слил изменения ветки branch_2 в ветку branch_1

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git switch branch_1
Switched to branch 'branch_1'
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git merge -m"merging 2 to 1" branch_2
Auto-merging 1.txt
CONFLICT (content): Merge conflict in 1.txt
Auto-merging 3.txt
CONFLICT (content): Merge conflict in 3.txt
Automatic merge failed; fix conflicts and then commit the result.
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 15 - Окно терминала после выполненных действий

17) Решил конфликт файла 1.txt в ручном режиме, а конфликт 3.txt используя команду git mergetool с помощью Meld.

Рисунок 16 – Отредактированный файл 1.txt

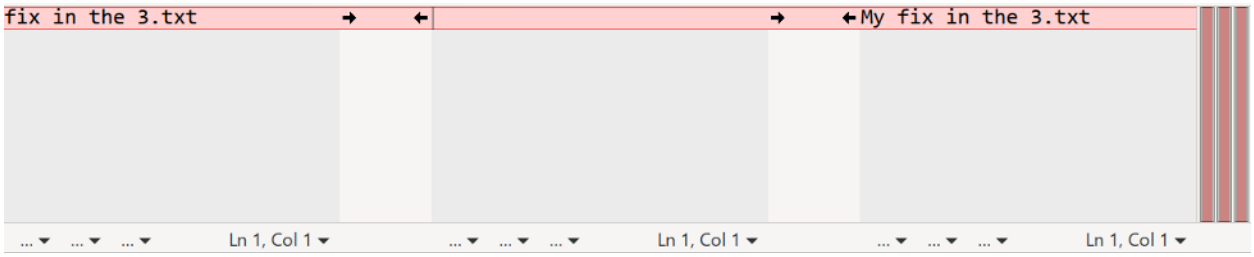

Рисунок 17 – Решение конфликта с помощью утилиты meld

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git push --set-upstream origin branch_1
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 8 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (14/14), 1.24 KiB | 1.24 MiB/s, done.
Total 14 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), done.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:      https://github.com/MelancholySeal/Python_LB3/pull/new/branch_1
remote:
To https://github.com/MelancholySeal/Python_LB3.git
 * [new branch]      branch_1 -> branch_1
branch 'branch_1' set up to track 'origin/branch_1'.
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 18 – Отправление ветки на удаленный репозиторий после решения конфликта

18) Создал branch_3 на удаленном репозитории

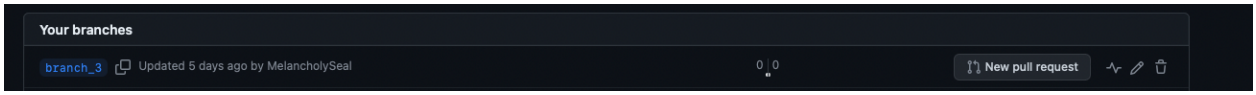

Рисунок 19 – Созданная ветка branch_3

19) Создал в локальном репозитории ветку отслеживания удаленной ветки branch_3

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git checkout --track origin/branch_3
branch 'branch_3' set up to track 'origin/branch_3'.
Switched to a new branch 'branch_3'
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 20 -Создание ветки отслеживания

20) Перешел на ветку branch_3 и добавил в файл 2.txt строку "the final fantasy in the 4.txt file"

```
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git switch branch_3
Already on 'branch_3'
Your branch is up to date with 'origin/branch_3'.
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % echo "the final fantasy in the 4.txt file" > 2.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 % git add 2.txt
(base) MelancholySeal@Kiras-MacBook-Air Python_LB3 %
```

Рисунок 21 – Изменение файла 2.txt

21) Выполнил перемещение ветки master на ветку branch_2

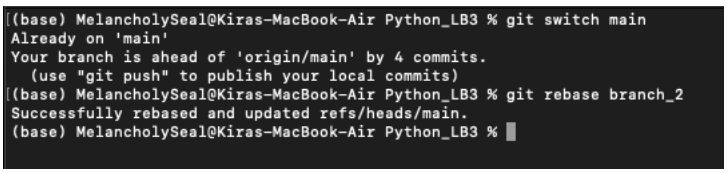

Рисунок 22 – Перемещение ветки main на branch_2

22) Отправил изменения в ветках main и branch_2 на удаленный репозиторий

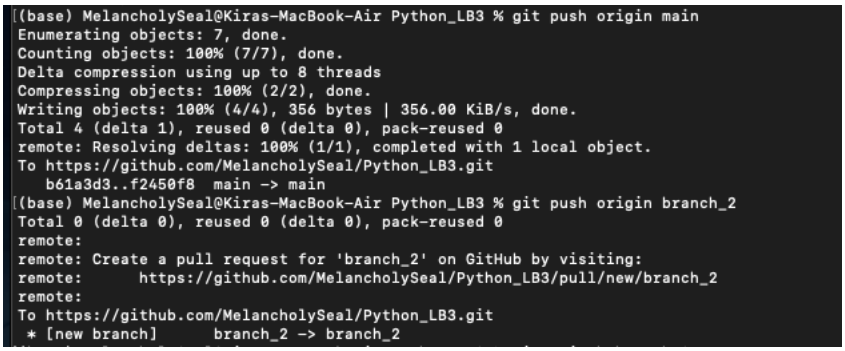

Рисунок 23 – Отправка изменений на удаленный репозиторий

Ответы на вопросы:

1) Что такое ветка?

Под веткой принято понимать независимую последовательность коммитов в хронологическом порядке. Однако конкретно в Git реализация ветки выполнена как указатель на последний коммит в рассматриваемой ветке. После создания ветки уже новый указатель ссылается на текущий коммит.

2) Что такое HEAD?

HEAD в Git-это указатель на текущую ссылку ветви, которая, в свою очередь, является указателем на последний сделанный вами коммит или последний коммит, который был извлечен из вашего рабочего каталога.

HEAD – это указатель на коммит в вашем репозитории, который станет родителем следующего коммита.

HEAD указывает на коммит, относительного которого будет создана рабочая копия во время операции checkout . Другими словами, когда вы переключаетесь с ветки на ветку, используя операцию checkout , то в вашем

репозитории указатель HEAD будет переключаться между последними коммитами выбираемых вами ветвей.

3) Способы создания веток.

1. Команда git branch: Создание новой ветки без переключения на нее;

2. команда git checkout -b: Создание и переключение на новую ветку одной командой;

3. создание веток в удаленных репозиториях (GitHub): веб-интерфейс позволяет создавать ветки и отправлять их в удаленный репозиторий.

4) Как узнать текущую ветку?

С помощью команд git branch и git status.

5) Как переключаться между ветками?

С помощью команд git checkout, git switch и git branch

6) Что такое удаленная ветка?

Удаленная ветка - это ветка, которая существует в удаленном репозитории и отслеживает состояние истории изменений в этом удаленном репозитории. Она может быть доступна для скачивания и обновления изменений между вашим локальным репозиторием и удаленным репозиторием. Удаленные ветки используются для совместной работы и синхронизации изменений между разными разработчиками и репозиториями.

7) Что такое ветка отслеживания?

Ветки слежения — это ссылки на определённое состояние удалённых веток. Это локальные ветки, которые нельзя перемещать; Git перемещает их автоматически при любой коммуникации с удаленным репозиторием, чтобы гарантировать точное соответствие с ним.

Ветка отслеживания - это локальная ветка в Git, которая непосредственно связана с удаленной веткой. Ветка отслеживания

автоматически отслеживает изменения в удаленной ветке и позволяет синхронизировать локальные изменения с удаленным репозиторием.

8) Как создать ветку отслеживания?

Для создания ветки отслеживания в Git, вы можете использовать команды git checkout или git switch с флагом -t (или --track).

9) Как отправить изменения из локальной ветки в удаленную ветку?

git push remote_name local_branch_name:remote_branch_name

remote_name: Имя удаленного репозитория, куда вы хотите отправить изменения (обычно это "origin").

local_branch_name: Имя вашей локальной ветки, из которой вы отправляете изменения.

remote_branch_name: Имя удаленной ветки, в которую вы хотите отправить изменения.

10) В чем отличие команд git fetch и git pull?

Команда git fetch загружает все изменения из удаленного репозитория в ваш локальный репозиторий, но не автоматически объединяет их с вашей текущей веткой. Это означает, что git fetch не изменяет вашу рабочую директорию и не создает новых коммитов в текущей ветке. Вместо этого он обновляет информацию о состоянии удаленных веток, которая хранится локально. После выполнения git fetch, вы можете решить, какие изменения объединить (если это необходимо) и когда.

Команда git pull также загружает изменения из удаленного репозитория в ваш локальный репозиторий, но, в отличие от git fetch, она автоматически пытается объединить эти изменения с вашей текущей веткой. git pull фактически объединяет изменения из удаленной ветки в вашу текущую ветку и создает новый коммит, если это необходимо. Это может привести к конфликтам слияния, если ваша текущая ветка и удаленная ветка имеют конфликтующие изменения.

11) Как удалить локальную и удаленную ветки?

Для удаления локальной ветки используется команда git branch -d с именем ветки, которую вы хотите удалить.

Удаление веток на удалённом сервере выполняется при помощи команды git push origin --delete

12) Изучить модель ветвления git-flow (использовать материалы статей https://www.atlassian.com/ru/git/tutorials/comparing-workflows/gitflowworkflow, https://habr.com/ru/post/106912/). Какие основные типы веток присуствуют в модели git-flow? Как организована работа светками в модели git-flow? В чем недостатки git-flow?

Модель git-flow предполагает следующие основные типы веток:

1. Main (Master) Branch**: Главная ветка, в которой хранится стабильная и готовая к продакшн версия продукта.
2. Develop Branch**: Ветка разработки, в которой объединяются новые функции и исправления из разных веток фичей. Здесь происходит основная разработка.
3. Feature Branches**: Ветки фичей, создаются для разработки новых функций. Каждая фича имеет свою собственную ветку, которая создается от ветки `develop` и после завершения фичи сливается обратно в `develop`.
4. Release Branches**: Ветки релизов, создаются перед выпуском новой версии. В них можно проводить финальное тестирование и подготовку к релизу. После завершения релиза ветка сливается как в `develop`, так и в `main` (для обновления стабильной версии).
5. Hotfix Branches**: Ветки исправлений, создаются для быстрого исправления критических ошибок в текущей стабильной версии (ветке `main`). После исправления ошибки ветка сливается как в `develop`, так и в `main`.

Работа с ветками в модели git-flow организована так:

1. Фичи создаются от `develop`.

2. Релизные ветки создаются перед выпуском новой версии и сливаются как в `main`, так и в `develop` после завершения тестирования.

3. Хотфиксы создаются от `main` для исправления критических ошибок и сливаются как в `main`, так и в `develop` после исправления.

Недостатки git-flow:

1. Сложность: Модель git-flow может быть слишком сложной для небольших проектов или команд, где требуется более простой подход к управлению ветками.

2. Замедление разработки: Создание множества дополнительных веток (фичей, релизов, хотфиксов) может замедлить процесс разработки и увеличить сложность слияния изменений.

3. Ветвление релизов: Ветки релизов могут стать сложными и требовать много усилий при долгосрочной разработке, особенно если между ними происходит много изменений.

4. Стандарт не всегда подходит: Модель git-flow не всегда идеально подходит для всех видов проектов и может потребовать адаптации к конкретным потребностям.

13) На прошлой лабораторной работе было задание выбрать одно из программных средств с GUI для работы с Git. Необходимо в рамках этого вопроса привести описание инструментов для работы с ветками Git, предоставляемых этим средством.

1. Создание веток: GitHub Desktop позволяет создавать новые локальные ветки на основе существующих веток в вашем репозитории. Вы можете указать имя и базовую ветку для новой ветки.

2. Переключение между ветками: Вы можете легко переключаться между локальными ветками с помощью интерфейса GitHub Desktop. Текущая активная ветка отображается в верхней части приложения.

3. Отслеживание удаленных веток: GitHub Desktop отображает доступные удаленные ветки для вашего репозитория. Вы можете создавать

локальные отслеживающие ветки для удаленных веток и синхронизировать изменения.

4. Просмотр истории веток: Инструмент предоставляет визуальное отображение истории изменений в ваших ветках. Вы можете просматривать коммиты и их связи между ветками.

5. Слияние веток: GitHub Desktop поддерживает слияние изменений из одной ветки в другую. Вы можете выполнить слияние локальных веток или изменений из удаленных веток.

6. Удаление веток:Вы можете удалять локальные ветки с помощью GitHub Desktop. Также есть возможность удаления удаленных веток (после подтверждения).

Вывод: лабораторная работа по исследованию базовых возможностей по работе с локальными и удаленными ветками Git позволила ознакомиться с важными аспектами управления ветками в системе контроля версий Git. Было изучено создание, переключение, удаление и слияние веток, а также поняли, как ветки используются для организации и совместной разработки в проектах.