МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет о лабораторной работе №1.3 по дисциплине основы программной инженерии

Выполнила:
Гробова Софья Кирилловна,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры
прикладной математики и
компьютерной безопасности,
Воронкин Р.А.

Отчет защищен с оценкой_____Дата защиты_____

Ставрополь, 2021 г.

ВЫПОЛНЕНИЕ:

1. Основы ветвления Git.

1.1 Добавление трёх файлов с помощью перезаписи коммитов (рис. 1).

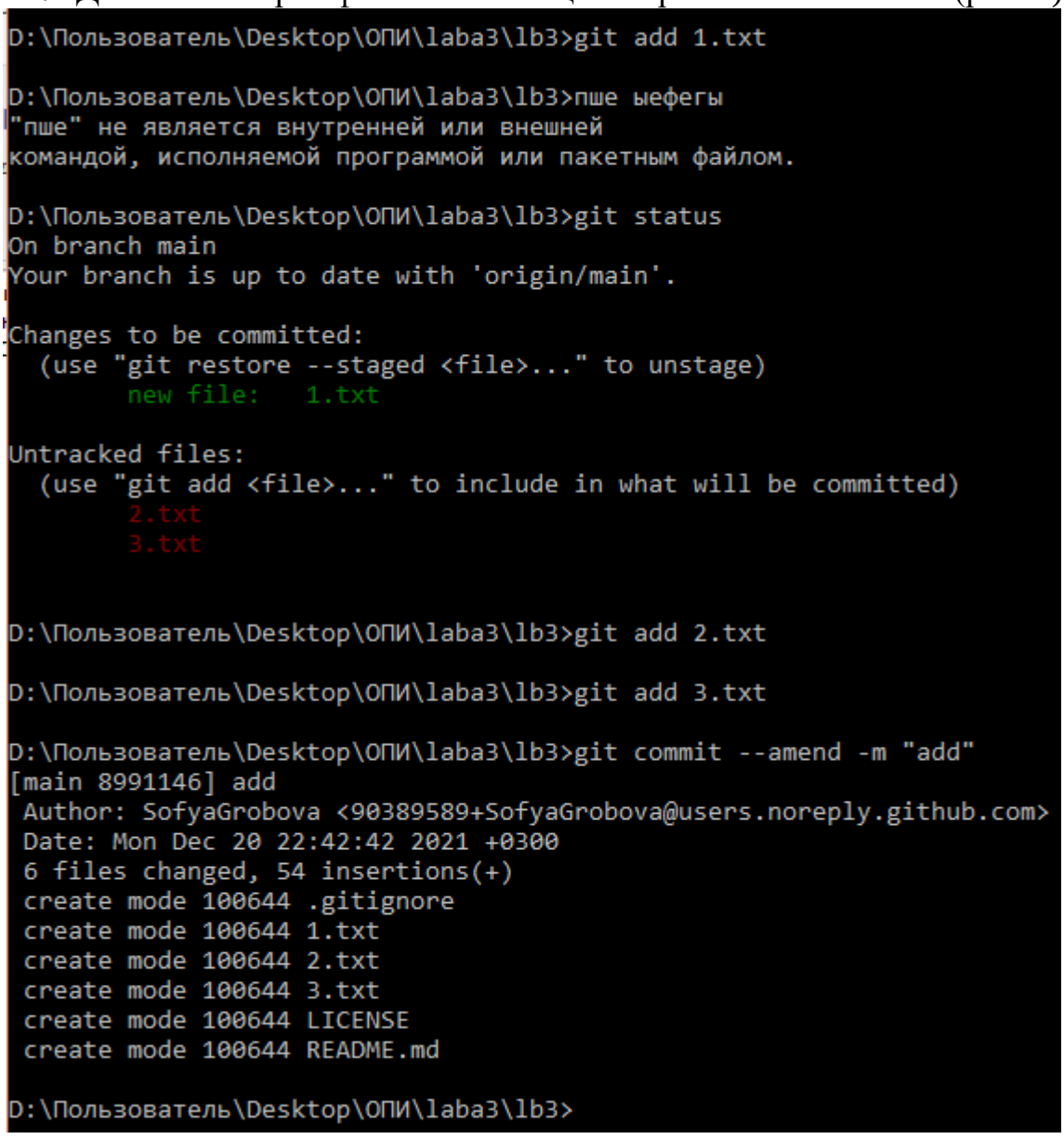

Рисунок 1 – Файлы «2.txt» и «3.txt» были внесены после коммита «1.txt» с помощью команды «git commit --amend»

1.2 Создание новой ветки (рис. 2).

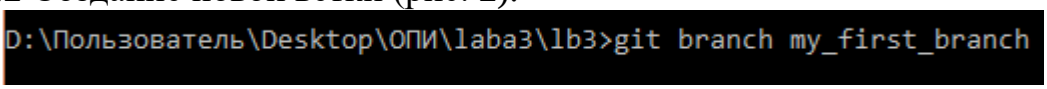

Рисунок 2 – Создание ветки «my_first_branch» с помощью команды «git branch»

1.3 Переход на новую ветку и создание файла (рис. 3).

```
D:\Пользователь\Desktop\ОПИ\laba3\lb3>git checkout my_first_branch
Switched to branch 'my_first_branch'

D:\Пользователь\Desktop\ОПИ\laba3\lb3>git status
On branch my_first_branch
nothing to commit, working tree clean

D:\Пользователь\Desktop\ОПИ\laba3\lb3>git add in_branch.txt

D:\Пользователь\Desktop\ОПИ\laba3\lb3>git commit -m "add in_branch"
[my_first_branch df32da1] add in_branch
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt

D:\Пользователь\Desktop\ОПИ\laba3\lb3>
```

Рисунок 3 – Создание файла «in_branch» в новой ветке

1.4 Создание новой ветки и автоматическое переключение на нее (рис.4).

```
D:\Пользователь\Desktop\ОПИ\laba3\lb3>git checkout -b new_branch
Switched to a new branch 'new_branch'
```

Рисунок 4 – На новую ветку можно перейти сращу после создания при помощи команды «git checkout -b»

1.5 Изменение файла в новой ветке (рис. 5).

```
D:\Пользователь\Desktop\ОПИ\laba3\lb3>git status
On branch new_branch
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:   1.txt

no changes added to commit (use "git add" and/or "git commit -a")

D:\Пользователь\Desktop\ОПИ\laba3\lb3>git add 1.txt

D:\Пользователь\Desktop\ОПИ\laba3\lb3>git commit -m "new row in the 1.txt file"
[new_branch bcfe503] new row in the 1.txt file
 1 file changed, 1 insertion(+)

D:\Пользователь\Desktop\ОПИ\laba3\lb3>
```

Рисунок 5 – Изменения в файле «1.txt» были закоммичены в ветке «new_branch»

1.6 Слияние главной ветки с побочными (рис. 6, 7).

```
D:\Пользователь\Desktop\ОПИ\laba3\lb3>git checkout main
Switched to branch 'main'
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

D:\Пользователь\Desktop\ОПИ\laba3\lb3>git merge my_first_branch
Updating 8991146..df32da1
Fast-forward
 in_branch.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt

D:\Пользователь\Desktop\ОПИ\laba3\lb3>git merge new_branch
Updating df32da1..bcfe503
Fast-forward
 1.txt | 1 +
 1 file changed, 1 insertion(+)

D:\Пользователь\Desktop\ОПИ\laba3\lb3>
```

f

Рисунок 6 – Слияние веток с помощью команды «git merge»

```
D:\Пользователь\Desktop\ОПИ\laba3\lb3>git branch -d my_first_branch
Deleted branch my_first_branch (was df32da1).

D:\Пользователь\Desktop\ОПИ\laba3\lb3>git branch -d new_branch
Deleted branch new_branch (was bcfe503).

D:\Пользователь\Desktop\ОПИ\laba3\lb3>
```

Рисунок 7 – Удаление ненужных веток

1.7 Конфликт слияния веток (рис. 8, 9 и 10).

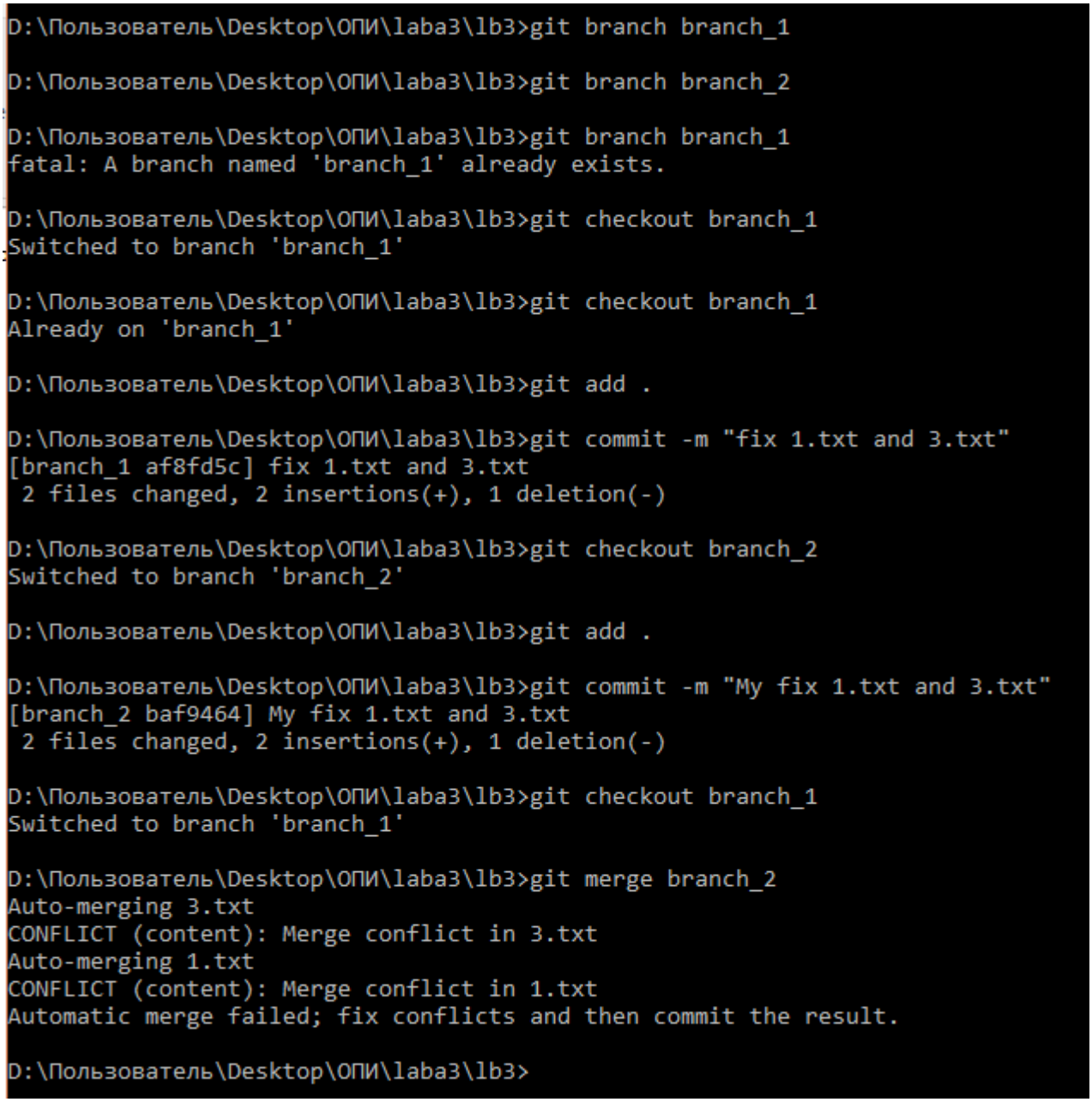

Рисунок 8 – Вывод консоли Git во время конфликта слияния

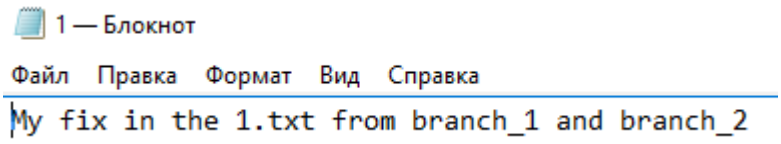

Рисунок 9 – Решение конфликта слияния в ручном режиме

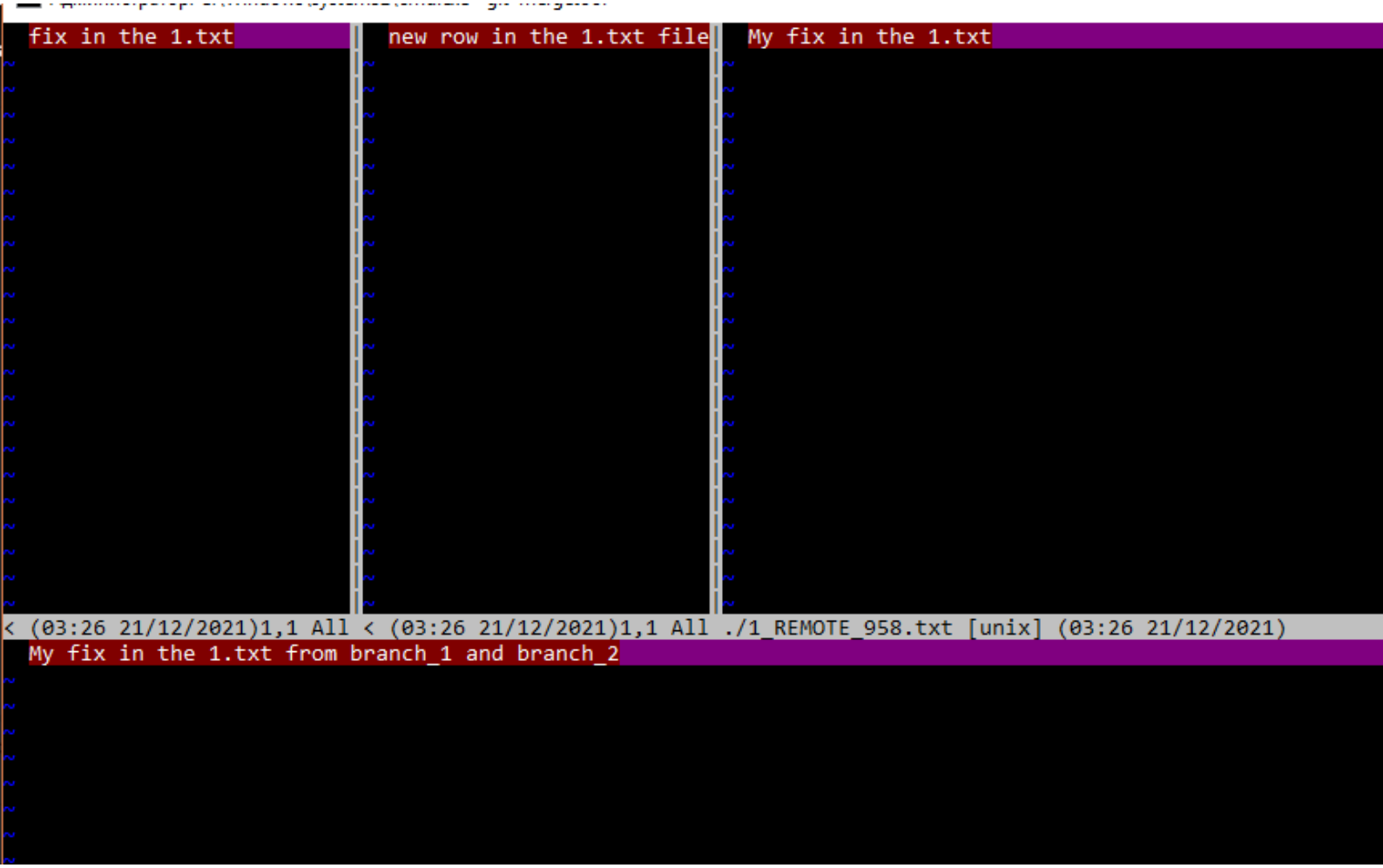

Рисунок 10 – Слияние с помощью «mergetool»

1.8 Отправка ветки в удаленный репозиторий (рис. 11).

```
D:\Пользователь\Desktop\ОПИ\laba3\lb3>git push origin branch_1
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 2 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (15/15), 1.87 KiB | 80.00 KiB/s, done.
Total 15 (delta 3), reused 3 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:      https://github.com/SofyaGrobova/lb3/pull/new/branch_1
remote:
To https://github.com/SofyaGrobova/lb3
 * [new branch]      branch_1 -> branch_1

D:\Пользователь\Desktop\ОПИ\laba3\lb3>
```

Рисунок 11 – Отправка ветки на GitHub с помощью команды «git push»

1.9 Создание удалённой ветки (рис. 12).

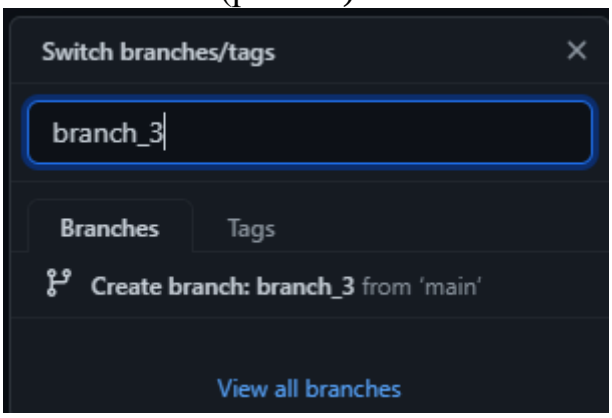

Рисунок 12 – Окно создания удалённой ветки

1.10 Создание ветки отслеживания (рис. 13).

```
D:\Пользователь\Desktop\ОПИ\laba3\lb3>git fetch --all
Fetching origin
From https://github.com/SofyaGrobova/lb3
 * [new branch]      branch_3   -> origin/branch_3

D:\Пользователь\Desktop\ОПИ\laba3\lb3>git checkout branch_3
error: you need to resolve your current index first
1.txt: needs merge
3.txt: needs merge
```

Рисунок 13 – Создание локальной ветки отслеживания «branch_3»

Ответы на вопросы

1. Ветка в Git — это простой перемещаемый указатель на один из коммитов. По умолчанию, имя основной ветки в Git — master.
2. HEAD – это указатель, задача которого ссылаться на определенный коммит в репозитории. Суть данного указателя можно попытаться объяснитьс разных сторон. Во-первых, HEAD – это указатель на коммит в вашем репозитории, который станет родителем следующего коммита. Во-вторых, HEAD указывает на коммит, относительного которого будет создана рабочаякопия во время операции checkout.

3. С помощью команды «git branch» или же «git checkout –b», в этомслучае создается новая ветка и указатель сразу перемещается на неё.

4. Если ввести команду git branch без параметров, то она выведет список всех веток и символом «*» пометит ветку, на которой вы находитесь.

5. С помощью команды «git checkout»

6. Удалённые ссылки — это ссылки (указатели) в ваших удалённых репозиториях, включая ветки, теги и так далее.

7. Ветки слежения — это локальные ветки, которые напрямую связаныс удалённой веткой. Если, находясь на ветке слежения, выполнить git pull, то Git уже будет знать с какого сервера получать данные и какую ветку использовать для слияния.

8. С помощью команды «git checkout --track».

9. С помощью команды «git push <remote> <branch>»

10. Команда «git fetch» получает с сервера все изменения, которых у вас ещё нет, но не будет изменять состояние вашей рабочей директории. Темне менее, существует команда «git pull», которая в большинстве случаев является командой «git fetch», за которой непосредственно следует команда «git merge». Если ваша ветка настроена определённым образом, или она явно установлена, или она была создана автоматически командами «clone» или «checkout», «git pull» определит сервер и ветку, за которыми следит ваша текущая ветка, получит данные с этого сервера и затем попытается слить удалённую ветку.

11. Удаленную ветку можно удалить с помощью команды «git push --delete», локальная ветка удаляется с помощью команды «git branch -d».

12. git-flow — это набор расширений git предоставляющий высокоуровневые операции над репозиторием для поддержки модели ветвления Vincent Driessen. В нём присутствуют такие ветки как «feature», «release» и «hotfix». Gitflow автоматизирует процессы слияния веток. Для начала использования необходимо установить gitflow и прописать команду «git flow init». Разработка новых фич начинается из ветки "develop". Для начала разработки фичи выполните:

git flow feature start MYFEATURE

Это действие создаёт новую ветку фичи, основанную на ветке "develop", и переключается на неё. Окончание разработки фичи. Это действиевыполняется так:

А. Слияние ветки MYFEATURE в "develop"

Б. Удаление ветки фичи
В. Переключение обратно на ветку "develop"
Г. git flow feature finish MYFEATURE

Для начала работы над релизом используйте команду git flow releaseОна создаёт ветку релиза, ответляя от ветки "develop".

git flow release start RELEASE [BASE]

При желании вы можете указать [BASE]-коммит в виде его хеша sha-1, чтобы начать релиз с него. Этот коммит должен принадлежать ветке "develop". Желательно сразу публиковать ветку релиза после создания, чтобыпозволить другим разработчиками выполнять коммиты в ветку релиза.
Это делается так же, как и при публикации фичи, с помощью команды:git

flow release publish RELEASE

Вы также можете отслеживать удалённый релиз с помощью командыgit flow

release track RELEASE

Завершение релиза — один из самых больших шагов в git-ветвлени.
При этом происходит несколько действий:

1) Ветка релиза сливается в ветку "master"
2) Релиз помечается тегом равным его имени
3) Ветка релиза сливается обратно в ветку "develop"
4) Ветка релиза удаляется git flow release

finish RELEASE

Не забудьте отправить изменения в тегах с помощью команды git push –tags.

Исправления нужны в том случае, когда нужно незамедлительно устранить нежелательное состояние продакшн-версии продукта. Она может ответвляться от соответствующего тега на ветке "master", который отмечает выпуск продакшн-версии.

Как и в случае с другими командами git flow, работа над исправлением начинается так:

git flow hotfix start VERSION [BASENAME]

Аргумент VERSION определяет имя нового, исправленного релиза.

При желании можно указать BASENAME-коммит, от которого произойдёт ответвление.

Когда исправление готово, оно сливается обратно в ветки "develop" и "master". Кроме того, коммит в ветке "master" помечается тегом с версией исправления.

git flow hotfix finish VERSION

Недостатки gitflow:

1. Git Flow может замедлять работу, когда приходится ревьювить большие пулл реквесты, когда вы пытаетесь выполнить итерациюбыстро.
2. Релизы сложно делать чаще, чем раз в неделю.
3. Большие функции могут потратить дни на мерж и резолв конфликтов и форсировать несколько циклов тестирования.
4. История проекта в гите имеет кучу merge commits и затрудняетпросмотр реальной работы.
5. Может быть проблематичным в CI/CD сценариях.

13.Пример с GItKraken

Теперь выполним подключение к удаленному репозиторию с клиента. Для этого в стартовом окне GitKraken выбираем последовательно Open Repo, Init, Local Only. В открывшемся окне нужно указать ссылку на удаленный репозиторий (из адресной строки браузера) и папку на компьютере, куда сохранятся файлы проекта. Если все сделано верно, содержимое репозиторияотобразится на клиенте. Теперь копия удаленного репозитория хранится локально на компьютере клиента. Отсюда можно работать с файлами проекта, и тогда изменения будутотправляться на сервер, в свою очередь клиент будет получать изменения, сделанные другими участниками проекта. Однако, для этого нужна авторизация на удаленном репозитории – она произойдет при попытке внесения любого первого изменения в проект при введении правильных учетных данных от удаленного репозитория. Разберем интерфейс этого окна. Слева можно переключаться между ветвями и репозиториями (локальным и удаленным), а также доступна навигация по тэгам. Кнопки на верхней панелипозволяют отменить или повторить последнее совершенное действие, сделатьпулл, пуш, создать новую ветку, создать стэш или освободить его (Стэш в гите – это черновик изменений. Когда работа над кодом еще не завершена, чтобы сделать коммит, но требуется переключиться на другую ветку – можно положить наработки в стэш, чтобы продолжить работу с ними позднее). Справа выводится список файлов, которые есть в репозитории. В правом верхнем углу расположены настройки учетной записи, кнопка поиска и вызоваменю настроек клиента. Создадим новую ветку и новый файл в ней. Для этого требуется нажать кнопку Branch, выбрать имя для новой ветки, после чего

появится сообщение об успешном создании новой ветки. Теперь в левом верхнем углу отображается в какой ветке мы находимся (был совершен авто-переход в только что созданную ветку). Создадим файл в новойветке. Для этого в области справа (там, где отображаются файлы проекта) нужно кликнуть правой кнопкой мыши и выбрать Create New File, после чего выбрать ему название. Откроется интерфейс редактирования файла. Раздел справа показывает изменения, подготовленные к коммиту, такназываемый стейдж. Иногда возникает ситуация, когда работа с одной частьюфайла завершена, а с другой – еще нет. Тогда через Стейдж можно закоммитить только необходимые изменения, а с остальными продолжить работу позже. Напишем простейший код в файл. Теперь нужно подготовить его к коммиту – поднять на стейдж кнопкой stage file. После чего останется только написать комментарий к коммиту и можно отправлять изменения в ветку. После чего совершаем сам коммит. Он появится на графе в центральной частиокна с комментарием и значком пользователя, его совершившего, а также на графе будет видно в какой ветке он совершен. Однако, изменения затронули только новую ветку. Чтобы изменения появились в главной ветви проекта, нужно сделать объединение (merge). Для этого нужно кликнуть правой кнопкой мыши по ветви secondary (текущая) и выбрать пункт меню «merge with master». Смотрим на итог применения объединения – изменился граф, теперь на нем видно, как ветка secondary слилась с основной, а справа отображаются файлы,добавленные в master – ранее эта ветка была пустой. Теперь отправим изменения на сервер. Для этого потребуется операция Push, совершаемая одноименной кнопкой на панели сверху. После непродолжительной загрузки высветится сообщение master pushed to origin, где под origin понимается состояние проекта на сервере (глобальное). Результат можем наблюдать в веб-интерфейсе GitLab – совершенные изменения и новый файл отображаются корректно.