МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ
УНИВЕРСИТЕТ
Институт Цифрового Развития
Кафедра инфокоммуникаций

ОТЧЁТ

по лабораторной работе №1

Дисциплина: «Языки программирования»

Выполнил: студент 2 курса
группы ИТС-б-о-20-1
Харченко Екатерина Владимировна
Проверил:
К.т.н., доцент кафедры инфокоммуникаций
Воронкин Роман Александрович

Работа защищена с оценкой: _____

Ставрополь, 2021 г.

ОСНОВЫ ВЕТВЛЕНИЯ GIT

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

Порядок выполнения работы:

1. Создаём общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT.

2. Создаем три файла: 1.txt, 2.txt, 3.txt.

3. Проиндексируем первый файл и сделаем коммит с комментарием "add 1.txt file".

```
C:\Users\Катя\lab2.1>git commit -m "add 1.txt file"
[main 83e1bda] add 1.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1.txt
```

4. Проиндексируем второй и третий файлы.

```
C:\Users\Катя\lab2.1>git add 2.txt

C:\Users\Катя\lab2.1>git commit -m "add 2.txt file"
[main 5e3a67d] add 2.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 2.txt

C:\Users\Катя\lab2.1>git add 3.txt

C:\Users\Катя\lab2.1>git commit -m "add 3.txt file"
[main 7d8d0aa] add 3.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 3.txt
```

5. Перезапишем уже сделанный коммит с новым комментарием "add 2.txt and 3.txt."

```
C:\Users\Катя\lab2.1>git commit --amend -m "add 2.txt and 3.txt."
[main 1c1a4fb] add 2.txt and 3.txt.
 Date: Tue Sep 14 18:47:04 2021 +0300
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 3.txt
```

6. Создадим новую ветку my_first_branch, и перейдем на неё.

```
C:\Users\Катя\lab2.1>git checkout -b my_first_branch
Switched to a new branch 'my_first_branch'

C:\Users\Катя\lab2.1>git checkout my_first_branch
Already on 'my_first_branch'
```

7. Создадим новый файл in_branch.txt, закоммитим изменения.

```
C:\Users\Катя\lab2.1>git add in_branch.txt

C:\Users\Катя\lab2.1>git commit -m "create my first branch"
[my_first_branch c906fd8] create my first branch
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt
```

8. Вернёмся на ветку main.

```
C:\Users\Катя\lab2.1>git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)
```

9. Создадим и сразу перейдем на ветку new_branch, при создании ветки переход производиться автоматически.

```
C:\Users\Катя\lab2.1>git checkout -b new_branch
Switched to a new branch 'new_branch'
```

10. Сделаем изменения в файле 1.txt, добавив строчку “new row in the 1.txt file”, закоммитим изменения.

```
C:\Users\Катя\lab2.1>git add 1.txt

C:\Users\Катя\lab2.1>git commit -m "new row in the 1.txt file"
[new_branch c4cc62c] new row in the 1.txt file
 1 file changed, 1 insertion(+)
```

11. Перейдем на ветку master и сольём ветки master и my_first_branch, после чего сольём ветки main и new_branch.

```
C:\Users\Катя\lab2.1>git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

C:\Users\Катя\lab2.1>git merge my_first_branch
Updating 1c1a4fb..c906fd8
Fast-forward
 in_branch.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt
```

```
C:\Users\Катя\lab2.1>git merge new_branch
Merge made by the 'recursive' strategy.
 1.txt | 1 +
 1 file changed, 1 insertion(+)
```

12. Удалим ветки my_first_branch и new_branch.

```
C:\Users\Катя\lab2.1>git branch -d my_first_branch
Deleted branch my_first_branch (was c906fd8).
```

```
C:\Users\Катя\lab2.1>git branch -d new_branch
Deleted branch new_branch (was c4cc62c).
```

13. Создадим ветки branch_1 и branch_2.

```
C:\Users\Катя\lab2.1>git checkout -b branch_1
Switched to a new branch 'branch_1'

C:\Users\Катя\lab2.1>git checkout -b branch_2
Switched to a new branch 'branch_2'
```

14. Перейдем на ветку branch_1 и изменим файл 1.txt, удалив все содержимое и добавив текст “fix in the 1.txt”, изменим файл 3.txt, удалив все содержимое и добавив текст “fix in the 3.txt”, закоммитим изменения.

```
C:\Users\Катя\lab2.1>git commit -m "fix in the 3.txt"
On branch branch_1
nothing to commit, working tree clean
```

15. Перейдем на ветку branch_2 и также изменим файл 1.txt, удалив все содержимое и добавив текст “My fix in the 1.txt”, изменим файл 3.txt, удалив все содержимое и добавив текст “My fix in the 3.txt”, закоммитим изменения.

```
C:\Users\Катя\lab2.1>git commit -m "My fix in the 3.txt"
[branch_2 f4aba31] My fix in the 3.txt
 2 files changed, 2 insertions(+), 1 deletion(-)
```

16. Сольём изменения ветки branch_2 в ветку branch_1.

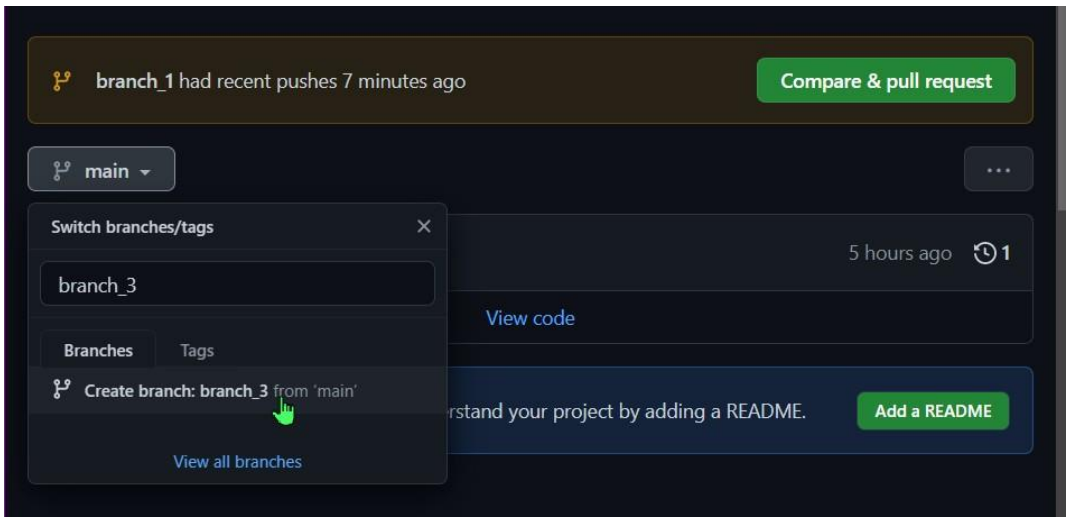
```
C:\Users\Катя\lab2.1>git checkout branch_1
Switched to branch 'branch_1'

C:\Users\Катя\lab2.1>git merge branch_2
Auto-merging 3.txt
CONFLICT (content): Merge conflict in 3.txt
Auto-merging 1.txt
CONFLICT (content): Merge conflict in 1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

17. Решим конфликт файла 1.txt в ручном режиме, а конфликт 3.txt используя команду git mergetool с помощью одной из доступных утилит, например, Meld.

18. Отправим ветку branch_1 на GitHub.

```
C:\Users\Катя\lab2.1>git push origin branch_1
info: please complete authentication in your browser...
Enumerating objects: 31, done.
Counting objects: 100% (31/31), done.
Delta compression using up to 8 threads
Compressing objects: 100% (24/24), done.
Writing objects: 100% (30/30), 2.78 KiB | 570.00 KiB/s, done.
Total 30 (delta 13), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (13/13), done.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:      https://github.com/ekaterina533/lab2.1/pull/new/branch_1
remote:
To https://github.com/ekaterina533/lab2.1
 * [new branch]      branch_1 -> branch_1
```

19. Создадим средствами GitHub удаленную ветку branch_3.



20. Создадим в локальном репозитории ветку отслеживания удаленной ветки branch_3.

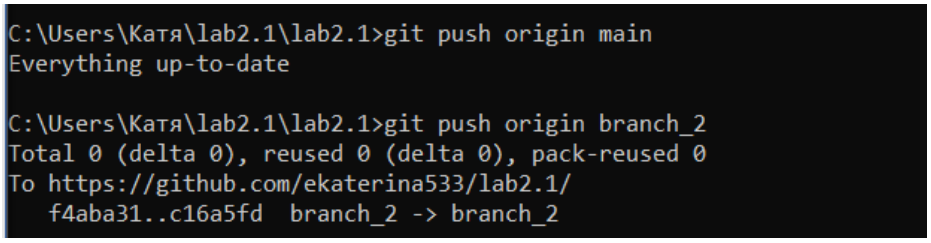
21. Перейдем на ветку branch_3 и добавим файл 2.txt строку "the final fantasy in the 4.txt file".

```
C:\Users\Катя\lab2.1\lab2.1>git checkout branch_3
Switched to a new branch 'branch_3'
Branch 'branch_3' set up to track remote branch 'branch_3' from 'origin'.

C:\Users\Катя\lab2.1\lab2.1>git add .

C:\Users\Катя\lab2.1\lab2.1>git commit -m "new row in the 2.txt file"
On branch branch_3
Your branch is up to date with 'origin/branch_3'.
```

22. Выполним перемещение ветки master на ветку branch_2.

```
C:\Users\Катя\lab2.1\lab2.1>git checkout branch_2
Switched to a new branch 'branch_2'
Branch 'branch_2' set up to track remote branch 'branch_2' from 'origin'.

C:\Users\Катя\lab2.1\lab2.1>git merge main
Updating f4aba31..c16a5fd
Fast-forward
 .1.txt.swp            | Bin 0 -> 12288 bytes
 .1_BASE_121.txt.swp   | Bin 0 -> 12288 bytes
 .1_LOCAL_121.txt.swp  | Bin 0 -> 12288 bytes
 .1_REMOTE_121.txt.swp | Bin 0 -> 12288 bytes
 1.txt                 |   6 +++++-
 1_BACKUP_121.txt      |   5 +++++
 1_BASE_121.txt        |   1 +
 1_LOCAL_121.txt       |   1 +
 1_REMOTE_121.txt      |   1 +
 3.txt                 |   6 +++++-
 4/txt                 |   1 +
 11 files changed, 19 insertions(+), 2 deletions(-)
 create mode 100644 .1.txt.swp
 create mode 100644 .1_BASE_121.txt.swp
 create mode 100644 .1_LOCAL_121.txt.swp
 create mode 100644 .1_REMOTE_121.txt.swp
 create mode 100644 1_BACKUP_121.txt
 create mode 100644 1_BASE_121.txt
 create mode 100644 1_LOCAL_121.txt
 create mode 100644 1_REMOTE_121.txt
 create mode 100644 4/txt
```

23. Отправим изменения веток master и branch_2 на GitHub.



Контрольные вопросы:

1. Что такое ветка?

Ветка в Git это подвижный указатель на один из коммитов. Обычно ветка указывает на последний коммит в цепочке коммитов. Ветка берет свое начало от какого-то одного коммита. Визуально это можно представить вот так.

2. Что такое HEAD?

HEAD – это указатель, задача которого ссылаться на определенный коммит в репозитории. 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 (о ветвлении в git будет рассказано в одной из ближайших статей), используя операцию checkout, то в вашем репозитории указатель HEAD будет переключаться между последними коммитами выбираемых вами ветвей.

3. Способы создания веток.

При создании ветки, всего лишь создаётся новый указатель для дальнейшего перемещения. Допустим вы хотите создать новую ветку с именем testing. Вы можете это сделать командой git branch:

4. Как узнать текущую ветку?

Чтобы узнать, какие ветки доступны и как называется текущая ветка, выполните команду git branch.

5. Как переключаться между ветками?

Если предположить, что ваш рабочий репозиторий уже содержит существующие ветки, вы можете переключаться между этими ветками с помощью команды git checkout

6. Что такое удаленная ветка?

Удалённые ссылки — это ссылки (указатели) в ваших удалённых репозиториях, включая ветки, теги и так далее. Полный список удалённых ссылок можно получить с помощью команды git ls-remote <remote> или команды git remote show <remote> для получения удалённых веток и дополнительной информации. Тем не менее, более распространенным способом является использование веток слежения.

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. Представляйте их как закладки для напоминания о том, где ветки в удалённых репозиториях находились во время последнего подключения к ним.

8. Как создать ветку отслеживания?

git branch --track style origin/style

git branch -a

git hist --max-count=2

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

С помощью команды git push

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

При использовании pull, git пытается сделать всё за вас. Он сливает любые внесённые коммиты в ветку, в которой вы сейчас работаете. Команда pull автоматически сливает коммиты, не давая вам сначала просмотреть их. Если вы не пристально следите за ветками, выполнение этой команды может привести к частым конфликтам.

При использовании fetch, git собирает все коммиты из целевой ветки, которых нет в текущей ветке, и сохраняет их в локальном репозитории. Однако он не сливает их в текущую ветку. Это особенно полезно, если вам нужно постоянно обновлять свой репозиторий, но вы работаете над функциональностью, неправильная реализация которой может негативно сказаться на проекте в целом.

11. Как удалить локальную и удаленную ветки?

Вы можете удалить ветку на удалённом сервере используя параметр --delete для команды git push

Чтобы удалить локальную ветку Git, вызовите команду git branch с параметром -d (--delete), за которым следует имя ветки.

12. Изучить модель ветвления git-flow. Какие основные типы веток присутствуют в модели git-flow? Как организована работа с ветками в модели git-flow? В чем недостатки git-flow?

Вместо использования одной ветки master, в этой модели используется две ветки для записи истории проекта. В ветке master хранится официальная история релиза, а ветка develop служит в качестве интеграционной ветки для новых функций. Также, удобно тегировать все коммиты в ветке master номером версии.

Git flow включает в себя мастер-ветку (*master*) и отдельную ветку для разработки (*develop*), а также дополнительные ветви для фич (*feature*), релизов (*release*) и патчей(*hotfix*).

По сравнению с моделью магистральной разработки, в Git-flow используется больше веток, каждая из которых существует дольше, а коммиты обычно крупнее. В соответствии с этой моделью разработчики создают функциональную ветку и откладывают ее слияние с главной магистральной веткой до завершения работы над функцией.

Под каждую новую функцию нужно выделить собственную ветку, которую можно отправить в центральный репозиторий для создания резервной копии или совместной работы команды. Ветки feature создаются не

на основе main, а на основе develop. Когда работа над функцией завершается, соответствующая ветка сливается с веткой develop.

Когда в ветке develop оказывается достаточно функций для выпуска (или приближается назначенная дата релиза), от ветки develop создается ветка release. Создание этой ветки запускает следующий цикл релиза, и с этого момента новые функции добавить больше нельзя — допускается лишь исправление багов, создание документации и решение других задач, связанных с релизом. Когда подготовка к поставке завершается, ветка release сливается с main и ей присваивается номер версии. Кроме того, нужно выполнить ее слияние с веткой develop, в которой с момента создания ветки релиза могли возникнуть изменения.

Первая проблема: авторам приходится использовать ветку *develop* вместо *master*, поскольку master зарезервирован для кода, который отправляется в продакшен. Существует сложившийся обычай называть рабочую ветвь по умолчанию *master*, и делать ответвления и слияния с ней. Большинство инструментов по умолчанию используют это название для основной ветки и по умолчанию выводят именно ее, и бывает неудобно постоянно переключаться вручную на другую ветку.

Вторая проблема процесса git flow – сложности, возникающие из-за веток для патчей и для релиза. Подобная структура может подойти некоторым организациям, но для абсолютного большинства она просто убийственно излишяя.

Вывод: в ходе лабораторной работы было исследованы базовые возможности по работе с локальными и удаленными ветками Git.