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 №3 по дисциплине «Основы программной инженерии»

Выполнил студент
2 курса, группы ПИЖ-б-о-20-1
Емельянова Я.А.
Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2021 г

1. Добавление файлов с помощью перезаписи коммитов.

```
C:\Users\Яна\lab3>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)

        deleted:    txt1.txt
        deleted:    txt2.txt
        deleted:    txt3.txt

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

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

C:\Users\Яна\lab3>git add 1.txt.txt

C:\Users\Яна\lab3>git commit -m "add 1.txt.txt file"
[main 808e1f0] add 1.txt.txt file
 1 file changed, 1 insertion(+)
 create mode 100644 1.txt.txt

C:\Users\Яна\lab3>git add 2.txt.txt

C:\Users\Яна\lab3>git add 3.txt.txt

C:\Users\Яна\lab3>git commit --amend -m "add 2.txt.txt and 3.txt.txt fi
les"
[main 42c801e] add 2.txt.txt and 3.txt.txt files
 Date: Tue Mar 8 23:49:47 2022 +0300
 3 files changed, 3 insertions(+)
 create mode 100644 1.txt.txt
 create mode 100644 2.txt.txt
 create mode 100644 3.txt.txt
```

Рис. 1 – перезапись коммитов с помощью команды –amend

2. Создание новой ветки и добавление файла.

```
C:\Users\Яна\lab3>git branch my_first_branch

C:\Users\Яна\lab3>git checkout my_first_branch
Switched to branch 'my_first_branch'
D       txt1.txt
D       txt2.txt
D       txt3.txt

C:\Users\Яна\lab3>git add branch.txt

C:\Users\Яна\lab3>git commit -m
error: switch `m' requires a value

C:\Users\Яна\lab3>git commit -m "add branch.txt"
[my_first_branch 1ae52f3] add branch.txt
 1 file changed, 1 insertion(+)
 create mode 100644 branch.txt
```

Рис. 2 – коммит файла в новой ветке

3. Создание новой ветки и изменение файла в ней

```
C:\Users\Яна\lab3>git checkout -b new_branch
Switched to a new branch 'new_branch'

C:\Users\Яна\lab3>git status
On branch new_branch
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)

        modified:   1.txt.txt
        deleted:    txt1.txt
        deleted:    txt2.txt
        deleted:    txt3.txt

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

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

C:\Users\Яна\lab3>git add 1.txt.txt

C:\Users\Яна\lab3>git commit -m "new row in the 1.txt.txt file"
[new_branch 30bd516] new row in the 1.txt.txt file
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Рис.3 – создание ветки и мгновенное переключение на неё с помощью команды “git checkout -b”, добавление файла в ветку

4. Слияние веток

```
C:\Users\Яна\lab3>git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

C:\Users\Яна\lab3>git merge my_first_branch
Updating 42c801e..1ae52f3
Fast-forward
 branch.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 branch.txt

C:\Users\Яна\lab3>git merge new_branch
Updating 1ae52f3..5a507fd
Fast-forward
 1.txt.txt   | 2 +-
 branch.docx | 0
 txt1.txt    | 1 -
 txt2.txt    | 1 -
 txt3.txt    | 1 -
 5 files changed, 1 insertion(+), 4 deletions(-)
 create mode 100644 branch.docx
 delete mode 100644 txt1.txt
 delete mode 100644 txt2.txt
 delete mode 100644 txt3.txt
```

Рис. 4 – слияние побочных веток с главной с помощью “git merge”

5. Удаление веток

```
C:\Users\Яна\lab3>git branch -d my_first_branch
Deleted branch my_first_branch (was 1ae52f3).

C:\Users\Яна\lab3>git branch -d new_branch
Deleted branch new_branch (was 5a507fd).
```

Рис. 5 – удаление слитых веток с помощью “git branch -d”

6. Конфликт слияния веток

```
C:\Users\Яна\lab3>git branch branch1

C:\Users\Яна\lab3>git branch branch2
```

Рис. 6.1 – создание двух веток

```
C:\Users\Яна\lab3>git status
On branch branch1
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)

        modified:   1.txt.txt
        deleted:    branch.docx

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

C:\Users\Яна\lab3>git add .

C:\Users\Яна\lab3>git commit -m "fix in 1.txt.txt"
[branch1 0ab277f] fix in 1.txt.txt
 2 files changed, 1 insertion(+)
 delete mode 100644 branch.docx
```

Рис. 6.2 – изменение файлов в ветке branch1

```
C:\Users\Яна\lab3>git checkout branch2
Switched to branch 'branch2'

C:\Users\Яна\lab3>git status
On branch branch2
nothing to commit, working tree clean

C:\Users\Яна\lab3>git status
On branch branch2
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.txt

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

C:\Users\Яна\lab3>git add .

C:\Users\Яна\lab3>git commit -m "my fix in 1.txt.txt"
[branch2 6959141] my fix in 1.txt.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Рис. 6.3 - изменение файлов в ветке branch2

```
C:\Users\Яна\lab3>git checkout branch1
Switched to branch 'branch1'

C:\Users\Яна\lab3>git merge branch2
Auto-merging 1.txt.txt
CONFLICT (content): Merge conflict in 1.txt.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Рис. 6.4 – конфликт слияния веток

7. Отправка ветки

```
C:\Users\Яна\lab3>git pushorigin branch1
git: 'pushorigin' is not a git command. See 'git --help'.

C:\Users\Яна\lab3>git push origin branch1
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 4 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (13/13), 1.12 KiB | 576.00 KiB/s, done.
Total 13 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), completed with 1 local object.
remote:
remote: Create a pull request for 'branch1' on GitHub by visiting:
remote:      https://github.com/astaffes/lab3/pull/new/branch1
remote:
To https://github.com/astaffes/lab3.git
 * [new branch]      branch1 -> branch1
```

Рис. 7 – отправка ветки на GitHub

8. Создание удалённой ветки

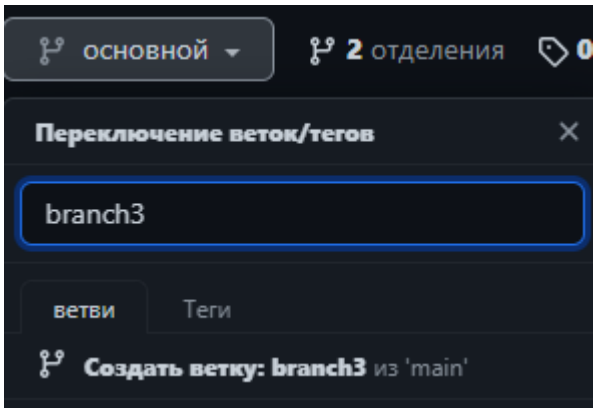

Рис. 8 – окно создания удалённой ветки

9. Создание ветки отслеживания

```
C:\Users\Яна\lab3>git branch -vv
* branch1 0ab277f fix in 1.txt.txt
  branch2 6959141 my fix in 1.txt.txt
  main    5a507fd [origin/main: ahead 4] new row in the 1.txt.txt file
```

Рис. 9 – ветка отслеживания для branch3

Ответы на вопросы:

1. Что такое ветка?

Ветка в Git — это простой перемещаемый указатель на один из коммитов. По умолчанию, имя основной ветки в Git — master.

2. Что такое HEAD?

HEAD – это указатель, задача которого ссылаться на определенный коммит в репозитории. Суть данного указателя можно попытаться объяснить с 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 рабочая копия во время операции checkout.

3. Способы создания веток.

С помощью команды «git branch» или же «git checkout –b» - в этом случае создается новая ветка и указатель сразу перемещается на неё.

4. Как узнать текущую ветку?

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

5. Как переключаться между 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.

7. Что такое ветка отслеживания?

Ветки слежения — это локальные ветки, которые напрямую связаны с 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. Как создать ветку отслеживания?

С помощью команды «git checkout --track».

9. Как отправить изменения из локальной ветки в удаленную? С помощью команды «git push <remote> <branch>»

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

Команда «git fetch» получает с сервера все изменения, которых у вас ещё нет, но не будет изменять состояние вашей рабочей директории. Команда «git pull», которая в большинстве случаев является командой «git

fetch», за которой непосредственно следует команда «git merge», определит сервер и ветку, за которыми следит ваша текущая ветка, получит данные с этого сервера и затем попытается слить удалённую ветку.

11. Как удалить локальную и удаленные ветки?

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

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

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 действие выполняется так:

1) Слияние ветки MYFEATURE в "develop"
2) Удаление ветки фичи
3) Переключение обратно на ветку "develop"
4) git flow feature finish MYFEATURE

Для начала работы над релизом используйте команду git flow release. Она 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, чтобы позволить 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 команды git 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 итерацию быстро.

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 затрудняет просмотр реальной работы.

5) Может быть проблематичным в CI/CD сценариях.

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.

Пример с GItKraken: для подключения удаленного репозитория 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 репозитория отобразится на клиенте.