

**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**

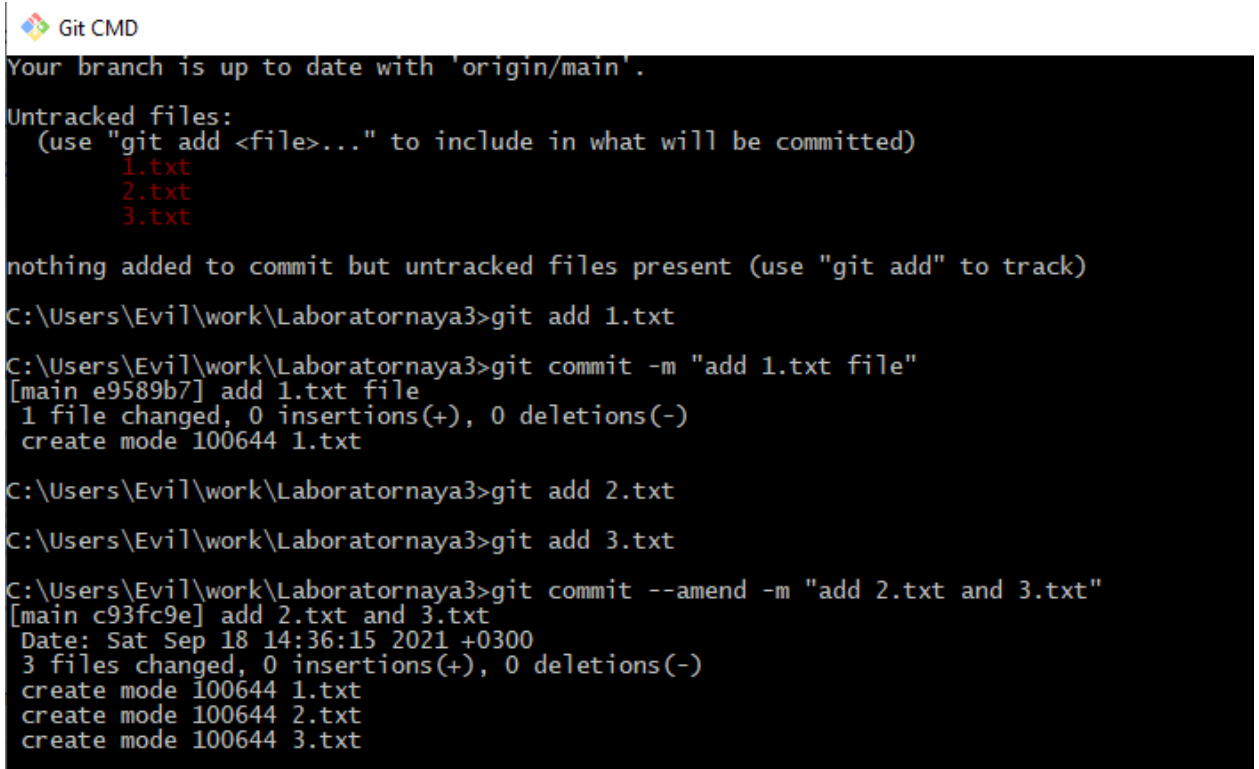
Выполнил:
Выходцев Егор Дмитриевич,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2021 г.

1. Основы ветвления Git.

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

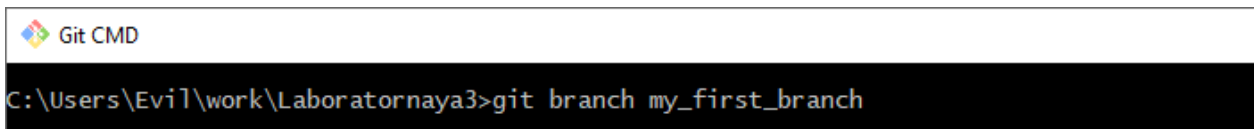


```
Git CMD
Your branch is up to date with 'origin/main'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        1.txt
        2.txt
        3.txt

nothing added to commit but untracked files present (use "git add" to track)
C:\Users\Evil\work\Laboratornaya3>git add 1.txt
C:\Users\Evil\work\Laboratornaya3>git commit -m "add 1.txt file"
[main e9589b7] add 1.txt file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 1.txt
C:\Users\Evil\work\Laboratornaya3>git add 2.txt
C:\Users\Evil\work\Laboratornaya3>git add 3.txt
C:\Users\Evil\work\Laboratornaya3>git commit --amend -m "add 2.txt and 3.txt"
[main c93fc9e] add 2.txt and 3.txt
Date: Sat Sep 18 14:36:15 2021 +0300
3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 1.txt
create mode 100644 2.txt
create mode 100644 3.txt
```

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

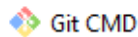
1.2 Создание новой ветки (рис. 2).



```
Git CMD
C:\Users\Evil\work\Laboratornaya3>git branch my_first_branch
```

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

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



```
C:\Users\Evil\work\Laboratornaya3>git checkout my_first_branch
Switched to branch 'my_first_branch'

C:\Users\Evil\work\Laboratornaya3>git status
On branch my_first_branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    in_branch.txt

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

C:\Users\Evil\work\Laboratornaya3>git add in_branch.txt

C:\Users\Evil\work\Laboratornaya3>git commit -m "add in_branch.txt"
[my_first_branch 640cf8b] add in_branch.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 in_branch.txt
```

Рисунок 3 – Создание файла «in_branch» в новой ветке

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



```
C:\Users\Evil\work\Laboratornaya3>git checkout -b new_branch
Switched to a new branch 'new_branch'
```

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

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

```
C:\Users\Evil\work\Laboratornaya3>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")

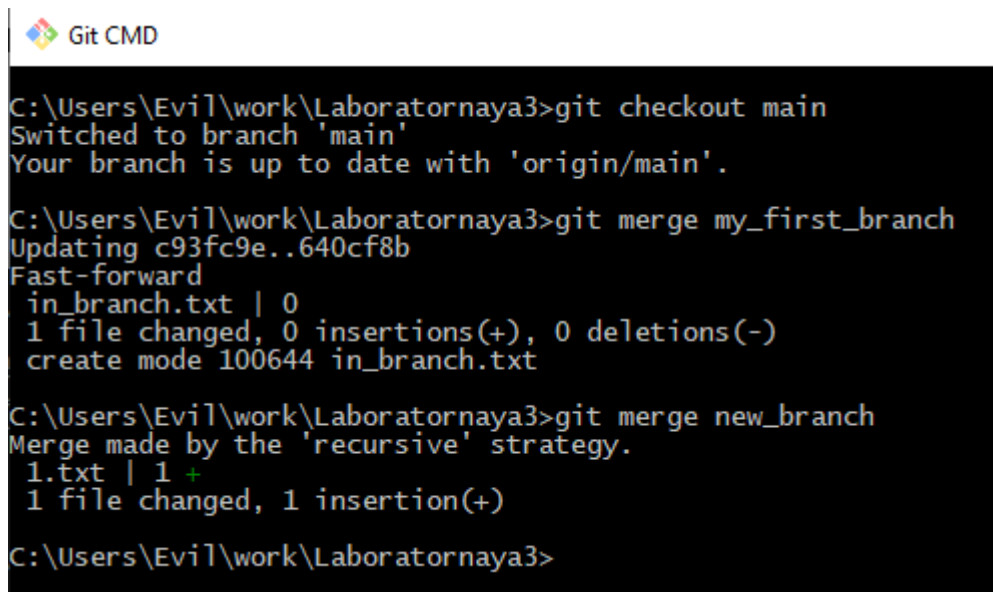
C:\Users\Evil\work\Laboratornaya3>git add 1.txt

C:\Users\Evil\work\Laboratornaya3>git commit -m "new row in 1.txt"
[new_branch 091042b] new row in 1.txt
1 file changed, 1 insertion(+)

C:\Users\Evil\work\Laboratornaya3>
```

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

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



```
Git CMD

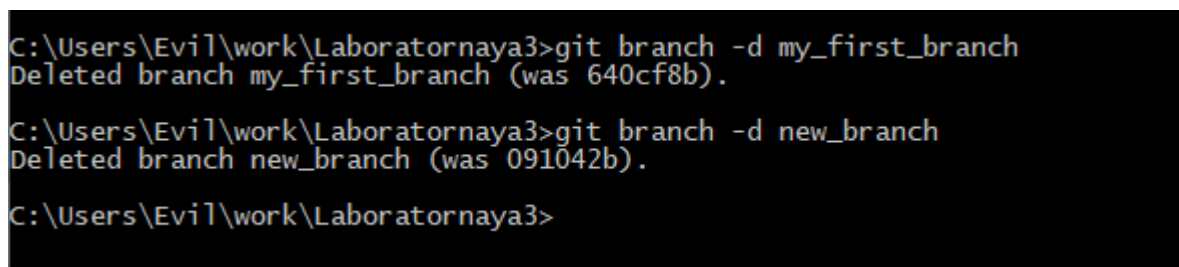
C:\Users\Evil\work\Laboratornaya3>git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

C:\Users\Evil\work\Laboratornaya3>git merge my_first_branch
Updating c93fc9e..640cf8b
Fast-forward
 in_branch.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt

C:\Users\Evil\work\Laboratornaya3>git merge new_branch
Merge made by the 'recursive' strategy.
 1.txt | 1 +
 1 file changed, 1 insertion(+)

C:\Users\Evil\work\Laboratornaya3>
```

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



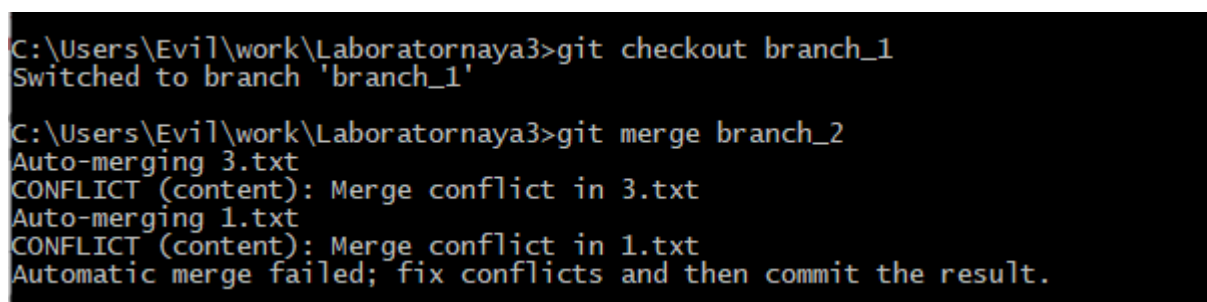
```
C:\Users\Evil\work\Laboratornaya3>git branch -d my_first_branch
Deleted branch my_first_branch (was 640cf8b).

C:\Users\Evil\work\Laboratornaya3>git branch -d new_branch
Deleted branch new_branch (was 091042b).

C:\Users\Evil\work\Laboratornaya3>
```

Рисунок 7 – Удаление ненужных веток

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



```
C:\Users\Evil\work\Laboratornaya3>git checkout branch_1
Switched to branch 'branch_1'

C:\Users\Evil\work\Laboratornaya3>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.
```

Рисунок 8 – Вывод консоли Git во время конфликта слияния

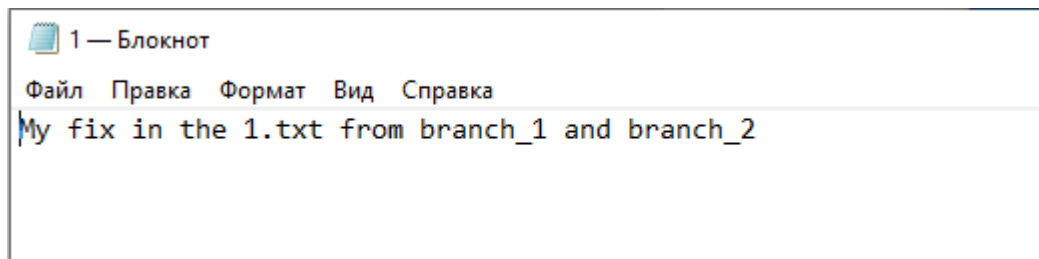


Рисунок 9 – Решение конфликта слияния в ручном режиме

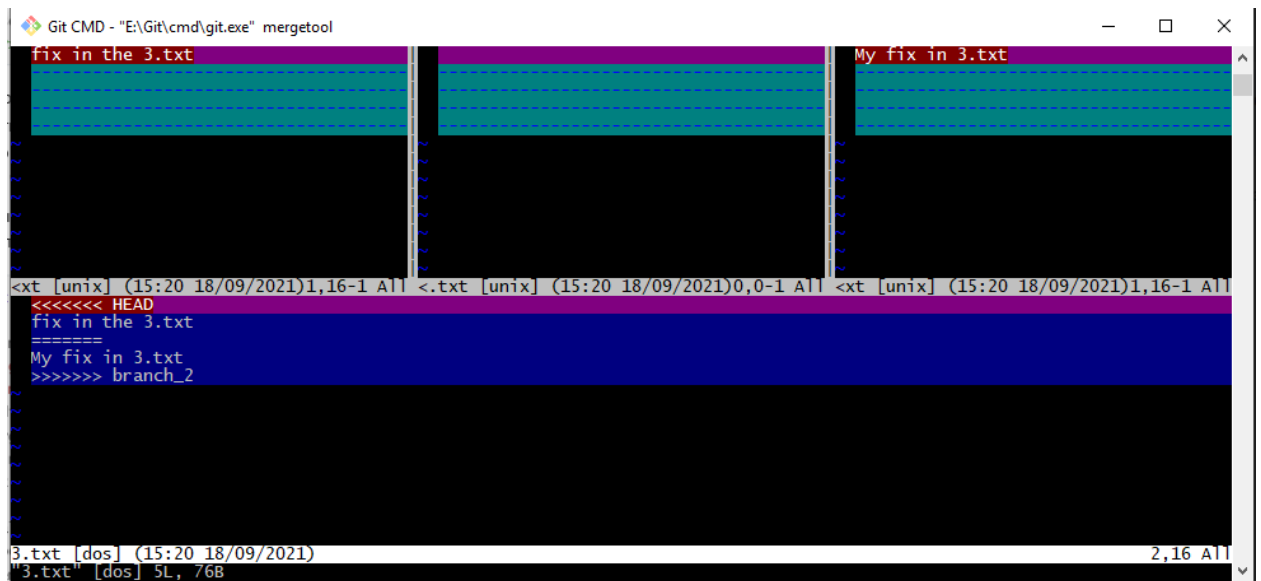


Рисунок 10 – Слияние с помощью «mergetool»

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

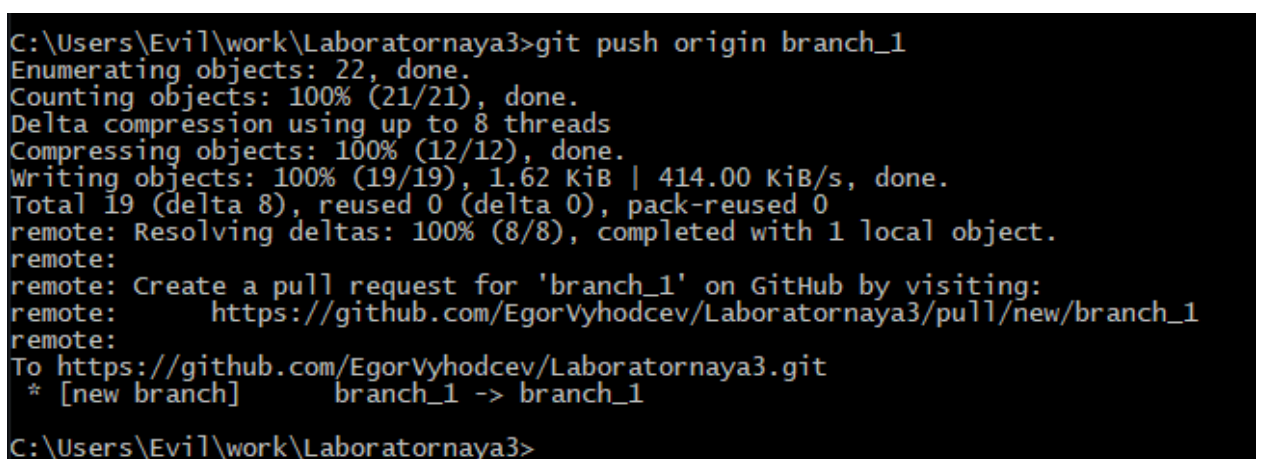


Рисунок 11 – Отправка ветки на GitHub с помощью команды «git push»

1.9 Создание удалённой ветки (рис. 12).

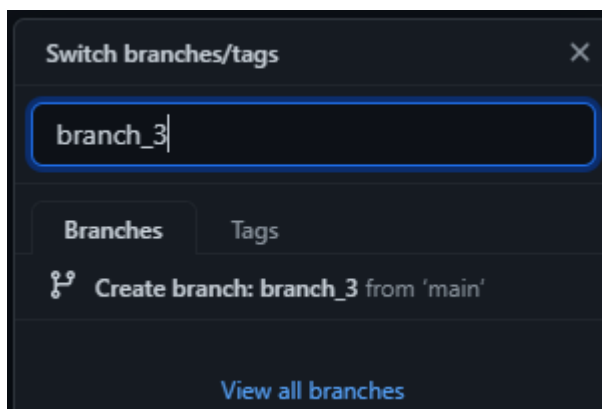


Рисунок 12 – Окно создания удалённой ветки

1.10 Создание ветки отслеживания (рис. 13).

```
C:\Users\Evil\work\Laboratornaya3>git fetch
From https://github.com/EgorVynodcev/Laboratornaya3
* [new branch]      branch_3    -> origin/branch_3

C:\Users\Evil\work\Laboratornaya3>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\Evil\work\Laboratornaya3>
```

Рисунок 13 – Создание локальной ветки отслеживания «branch_3»

1.11 Перебазирование веток (рис. 14).

```
C:\Users\Evil\work\Laboratornaya3>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\Evil\work\Laboratornaya3>git rebase branch_2
Successfully rebased and updated refs/heads/main.
```

Рисунок 14 – Перебазирование ветки «main» на «branch_2»

2. 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 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, относительно которого будет создана рабочая копия во время операции checkout.

3. С помощью команды «git branch» или же «git checkout -b», в этом случае 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 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. 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. Тем не 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. Удалённую 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 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, ответвляя 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 разработчикам 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-ветвлении. 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.

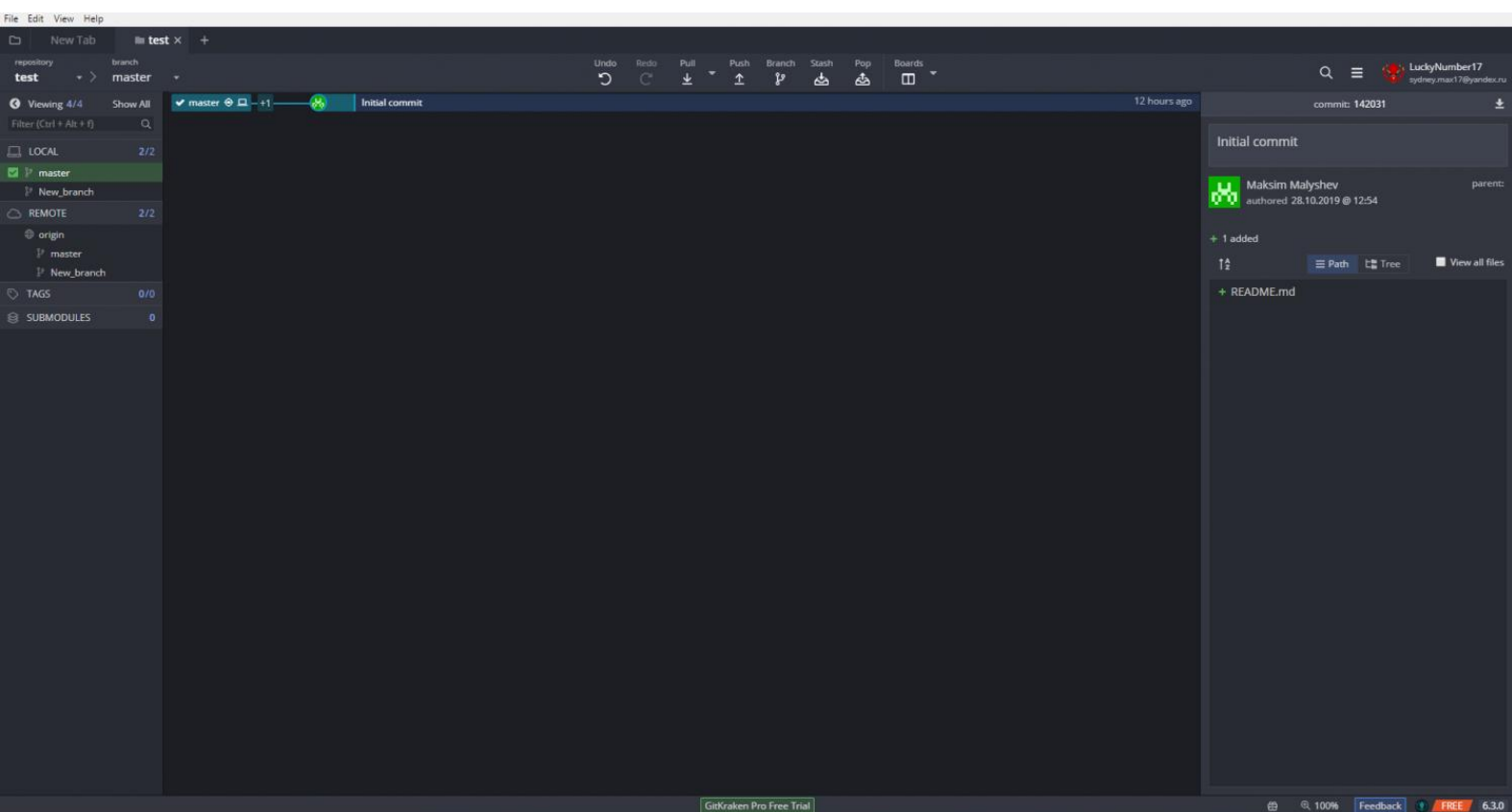
```
git flow hotfix finish VERSION
```

Недостатки gitflow:

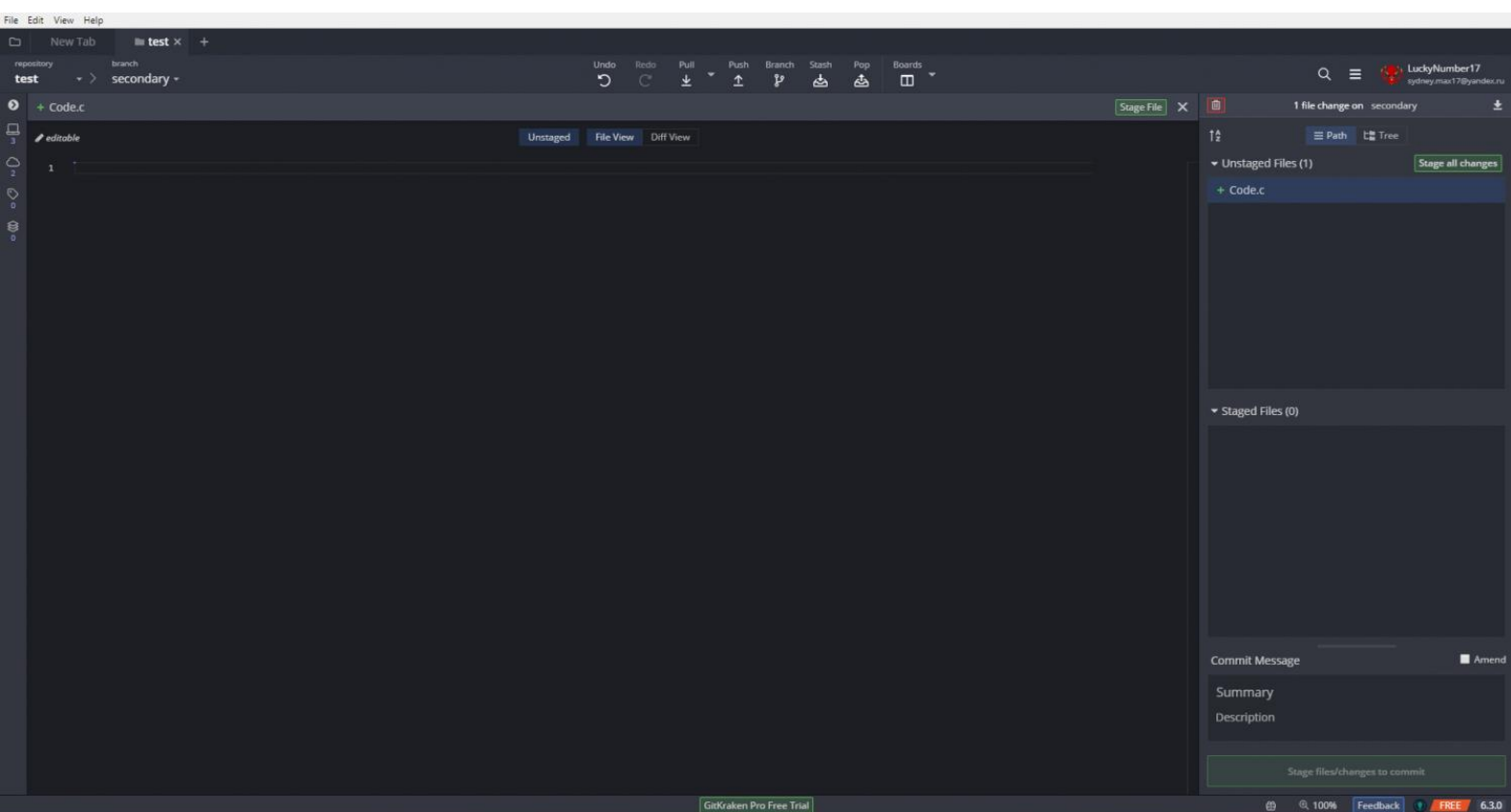
1. Git Flow может замедлять работу, когда приходится ревьюить большие пулл реквесты, когда вы пытаетесь выполнить итерацию быстро.
2. Релизы сложно делать чаще, чем раз в неделю.
3. Большие функции могут потратить дни на мерж и резолв конфликтов и форсировать несколько циклов тестирования.
4. История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

13. Пример с GitKraken

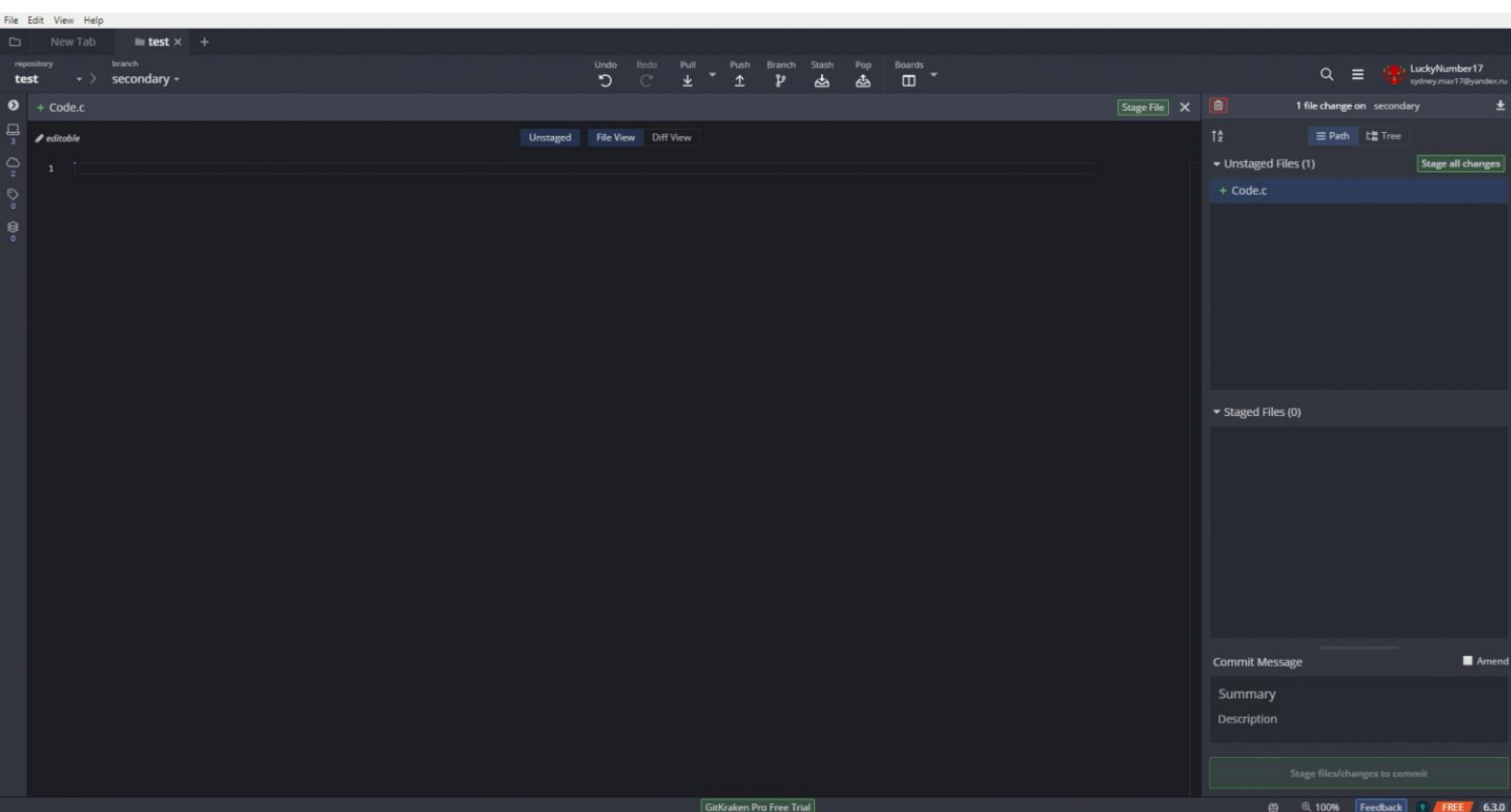
Теперь выполним подключение к удаленному репозиторию с клиента. Для этого в стартовом окне GitKraken выбираем последовательно Open Repo, Init, Local Only. В открывшемся окне нужно указать ссылку на удаленный репозиторий (из адресной строки браузера) и папку на компьютере, куда сохранятся файлы проекта. Если все сделано верно, содержимое репозитория отобразится на клиенте.



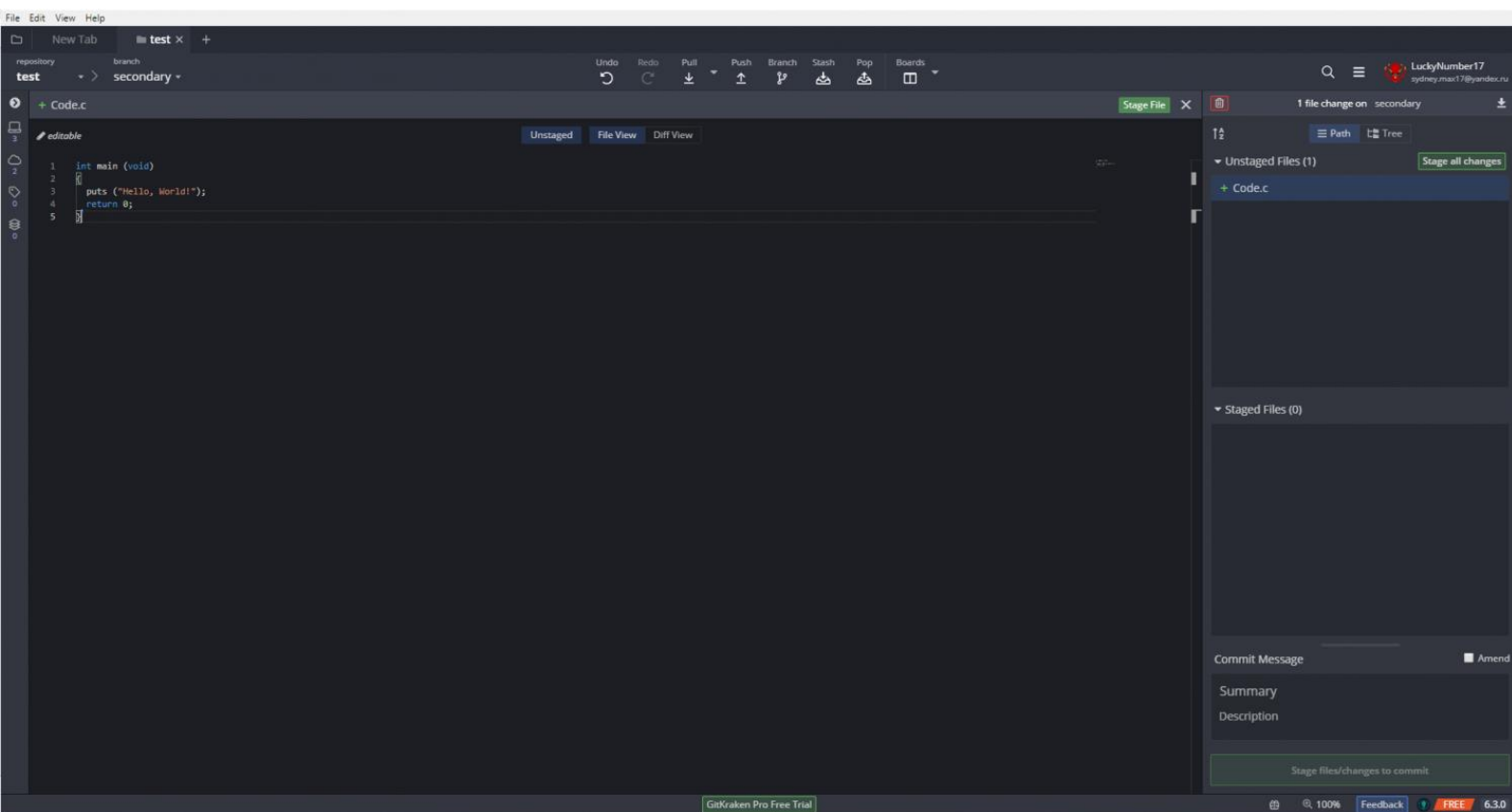
Теперь копия удаленного репозитория хранится локально на компьютере клиента. Отсюда можно работать с файлами проекта, и тогда изменения будут отправляться на сервер, в свою очередь клиент будет получать изменения, сделанные другими участниками проекта. Однако, для этого нужна авторизация на удаленном 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 панели позволяют отменить или повторить последнее совершенное действие, сделать пулл, 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 вызова меню 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.



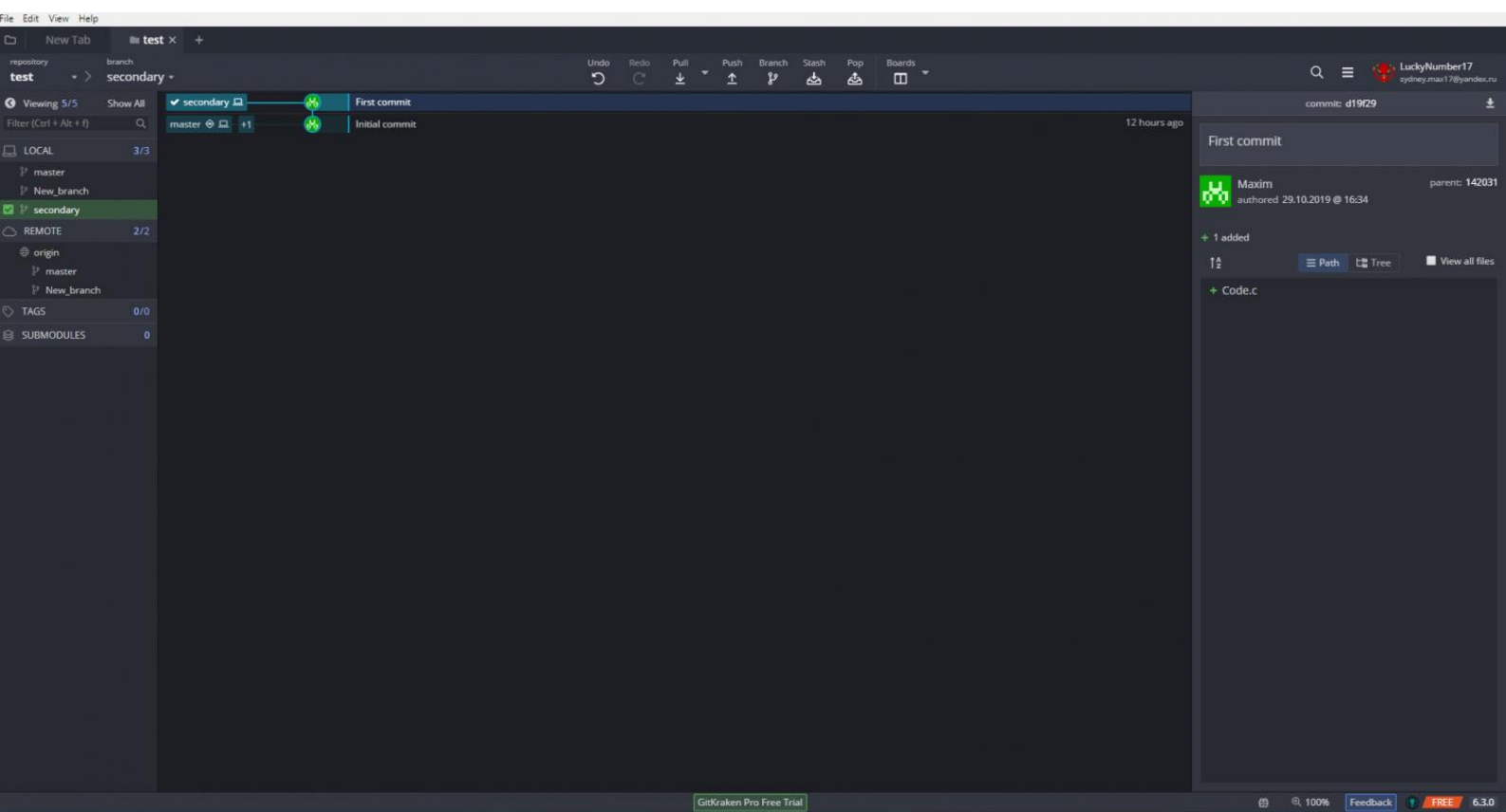
Теперь в левом верхнем углу отображается в какой ветке мы находимся (был совершен авто-переход в только что созданную ветку). Создадим файл в новой ветке. Для этого в области справа (там, где отображаются файлы проекта) нужно кликнуть правой кнопкой мыши и выбрать Create New File, после чего выбрать ему название. Откроется интерфейс редактирования файла.



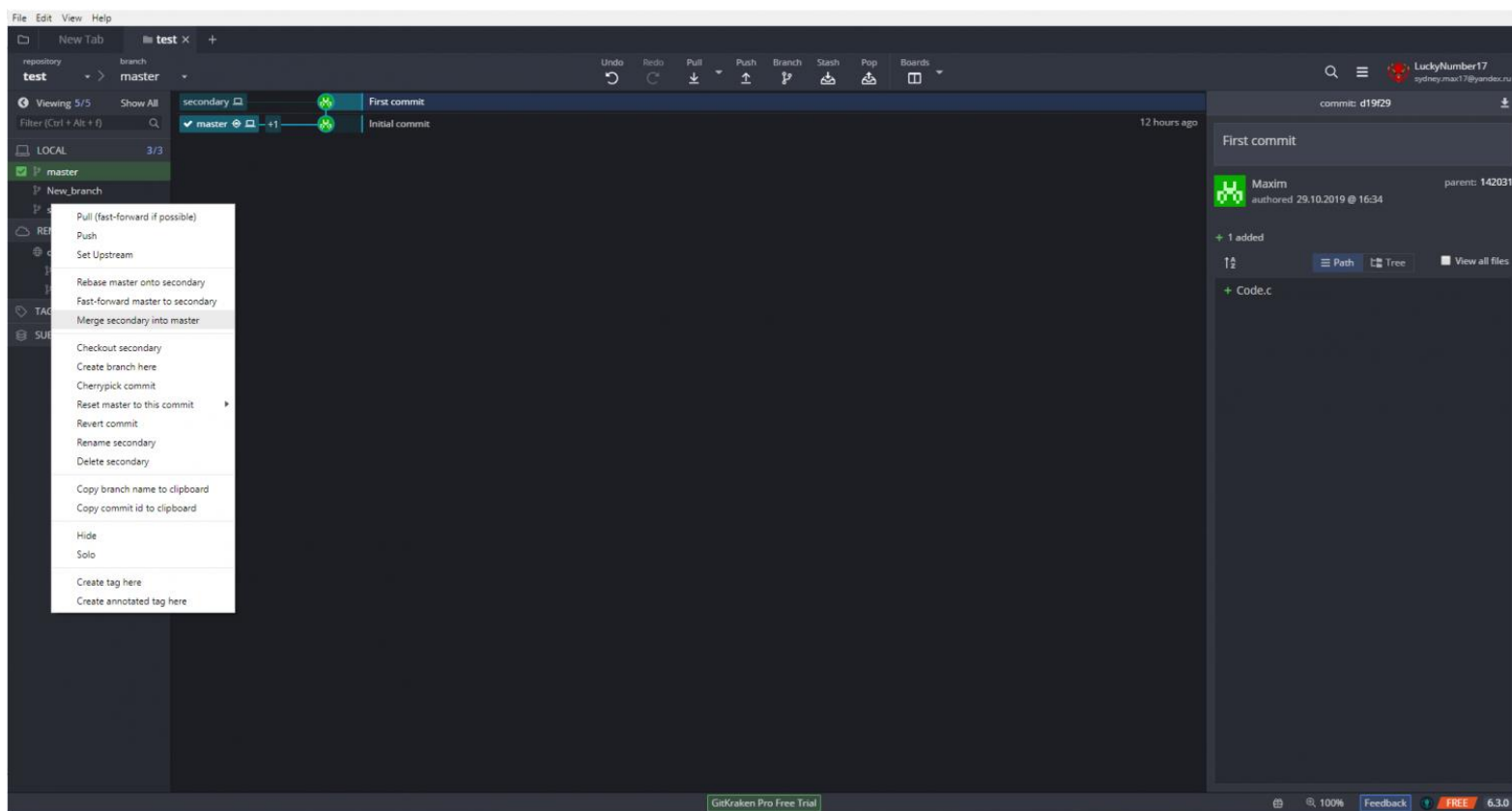
Раздел справа показывает изменения, подготовленные к коммиту, так называемый стейдж. Иногда возникает ситуация, когда работа с одной частью файла завершена, а с другой – еще нет. Тогда через Стейдж можно закоммитить только необходимые изменения, а с остальными продолжить работу позже. Напишем простейший 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.



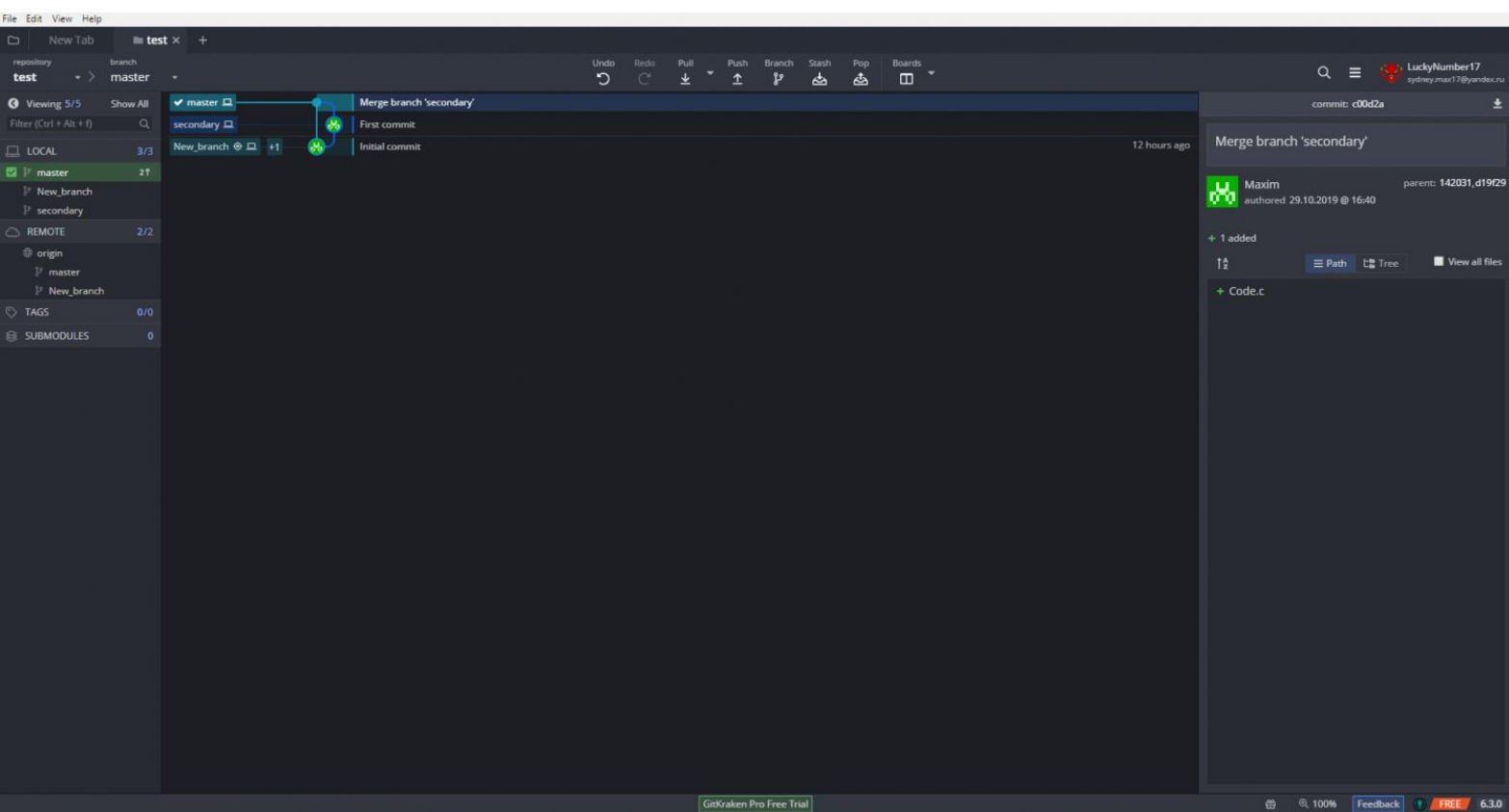
После чего совершаем сам коммит. Он появится на графе в центральной части окна с комментарием и значком пользователя, его совершившего, а также на графе будет видно в какой ветке он совершен.



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.