

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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
дисциплины
«Основы кроссплатформенного программирования»
Вариант

Выполнил:
Ляш Денис Александрович
2 курс, группа ИТС-б-о-23-1,
11.03.02 «Инфокоммуникационные
технологии и системы связи», очная
форма обучения

(подпись)

Проверил:
Доцент департамента цифровых,
робототехнических систем и
электроники Воронкин Р.А.

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

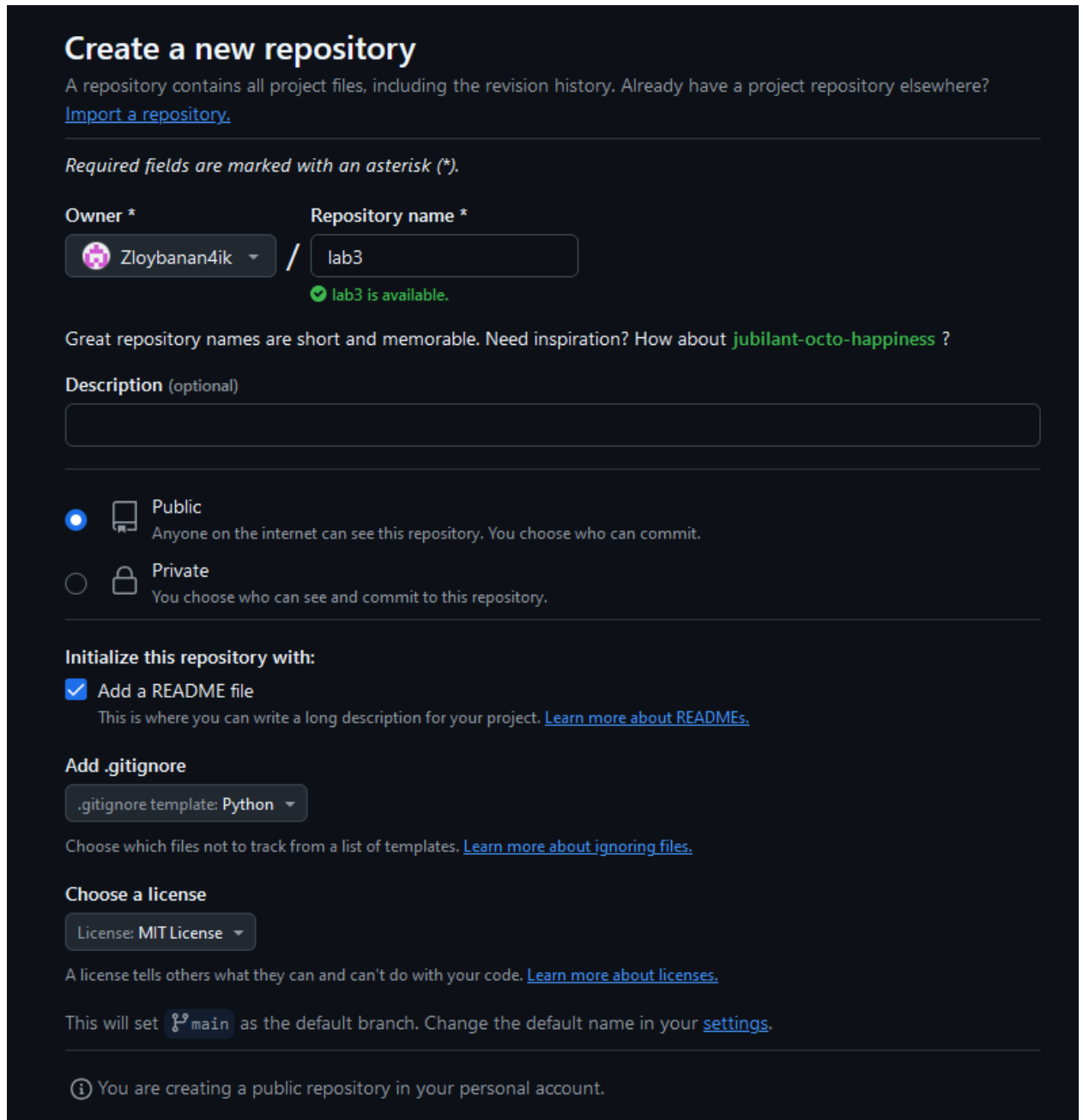
Ставрополь, 2024 г.

Тема: Основы языка Python

Цель работы: исследование процесса установки и базовых возможностей языка Python

Порядок выполнения работы:

1. Изучил теоретический материал работы.
2. Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и выбранный язык программирования.




Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*


Owner * / Repository name *


 Zloybanan4ik / lab3

✔ lab3 is available.

Great repository names are short and memorable. Need inspiration? How about [jubilant-octo-happiness](#) ?

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: **Python**

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **MIT License**

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).


 You are creating a public repository in your personal account.

Рисунок 1. Создание репозитория

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

Имя	Дата изменения	Тип	Размер
.git	29.12.2024 16:43	Папка с файлами	
.gitignore	29.12.2024 16:43	Текстовый докум...	4 КБ
1.txt	29.12.2024 16:43	Текстовый докум...	0 КБ
2.txt	29.12.2024 16:43	Текстовый докум...	0 КБ
3.txt	29.12.2024 16:43	Текстовый докум...	0 КБ
LICENSE	29.12.2024 16:43	Файл	2 КБ
README.md	29.12.2024 16:43	Исходный файл ...	1 КБ

Рисунок 2. Создание файлов

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

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace
$ cd lab3

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git add 1.txt

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git commit -m "add 1.txt"
[main 6c50de1] add 1.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 1.txt

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git add 2.txt

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git commit -m "add 2.txt"
[main fa7e6ab] add 2.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 2.txt

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git add 3.txt

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git commit -m "add 3.txt"
[main 9fc9765] add 3.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 3.txt
```

Рисунок 3. Добавление коммита с файлом

5. Проиндексировал второй и третий файлы.

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git commit --amend -m "add 2.txt and 3.txt"
[main b9a242b] add 2.txt and 3.txt
Date: Sun Dec 29 16:47:48 2024 +0300
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 3.txt
```

Рисунок 4. Изменение коммита

6. Перезаписал уже сделанный коммит с новым комментарием "add 2.txt and 3.txt."

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git branch my_first_branch
```

Рисунок 5. Создание ветки

7. Создал новую ветку my_first_branch.

8. Перейти на ветку и создал новый файл in_branch.txt, закоммитил изменения.

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git checkout my_first_branch
Switched to branch 'my_first_branch'
```

Рисунок 6. Переключение на новую ветку

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (my_first_branch)
$ git add in_branch.txt

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (my_first_branch)
$ git commit -m "add file in new branch"
[my_first_branch 893562a] add file in new branch
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 in_branch.txt
```

Рисунок 7. Создание коммита на новой ветке

9. Вернулся на ветку main.

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (my_first_branch)
$ 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)
\
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
```

Рисунок 8. Переключение на главную ветку

10. Создал и сразу перешел на ветку new_branch.

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git branch new_branch

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git checkout new_branch
Switched to branch 'new_branch'
```

Рисунок 9. Создание и переключение на новую ветку

11. Сделал изменения в файле 1.txt, добавил строчку “new row in the 1.txt file”, закоммитил изменения.

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (new_branch)
$ git add 1.txt

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (new_branch)
$ git commit -m "change file 1.txt"
[new_branch f8f8d29] change file 1.txt
1 file changed, 1 insertion(+)
```

Рисунок 10. Изменение файла

12. Перешел на ветку main и слил ветки main и my_first_branch, после чего слил ветки main и new_branch.

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git merge my_first_branch
Updating b9a242b..893562a
Fast-forward
 in_branch.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git merge new_branch
Merge made by the 'ort' strategy.
 1.txt | 1 +
 1 file changed, 1 insertion(+)
```

Рисунок 11. Слияние веток

13. Удалил ветки my_first_branch и new_branch.

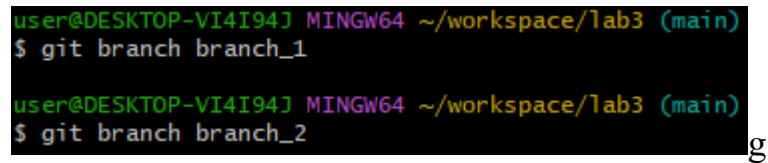
```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git branch -D my_first_Branch
Deleted branch my_first_Branch (was 893562a).
```

Рисунок 12. Удаление ветки my_first_branch

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git branch -D new_branch
Deleted branch new_branch (was f8f8d29).
```

Рисунок 13. Удаление ветки new_branch

14. Создал ветки branch_1 и branch_2.

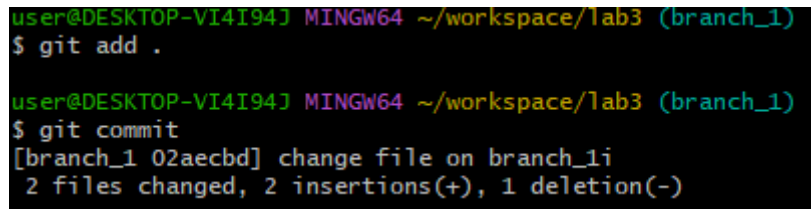
A terminal window showing the creation of two Git branches. The prompt is 'user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)'. The first command is '\$ git branch branch_1'. The second command is '\$ git branch branch_2'.

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git branch branch_1

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git branch branch_2
```

Рисунок 14. Создание веток

15. Перешел на ветку branch_1 и изменил файл 1.txt, удалил все содержимое и добавил текст “fix in the 1.txt”, изменил файл 3.txt, удалил все содержимое и добавил текст “fix in the 3.txt”, закоммитил изменения.

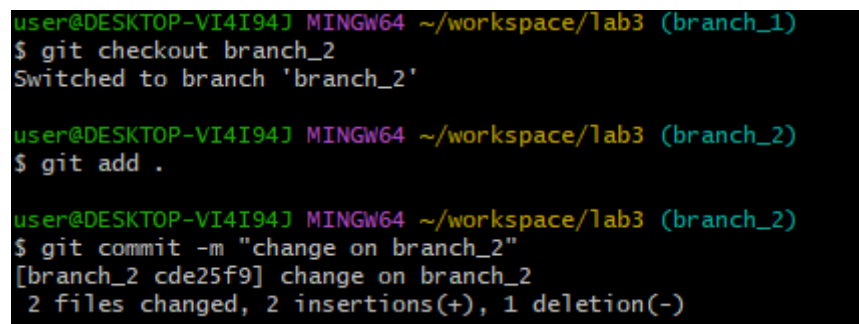
A terminal window showing the process of committing changes on the branch_1. The prompt is 'user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1)'. The commands are '\$ git add .' and '\$ git commit'. The output shows the commit message '[branch_1 02aecbd] change file on branch_1' and '2 files changed, 2 insertions(+), 1 deletion(-)'.

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1)
$ git add .

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1)
$ git commit
[branch_1 02aecbd] change file on branch_1
2 files changed, 2 insertions(+), 1 deletion(-)
```

Рисунок 15. Изменение файла на ветке branch_1

16. Перешел на ветку branch_2 и также изменил файл 1.txt, удалил все содержимое и добавил текст “My fix in the 1.txt”, изменил файл 3.txt, удалил все содержимое и добавил текст “My fix in the 3.txt”, закоммитил изменения.

A terminal window showing the process of committing changes on the branch_2. The prompt is 'user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1)'. The first command is '\$ git checkout branch_2', which switches to 'branch_2'. The second command is '\$ git add .' and the third is '\$ git commit -m "change on branch_2"'. The output shows the commit message '[branch_2 cde25f9] change on branch_2' and '2 files changed, 2 insertions(+), 1 deletion(-)'.

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1)
$ git checkout branch_2
Switched to branch 'branch_2'

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_2)
$ git add .

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_2)
$ git commit -m "change on branch_2"
[branch_2 cde25f9] change on branch_2
2 files changed, 2 insertions(+), 1 deletion(-)
```

Рисунок 16. Изменение файла на ветке branch_2

17. Слил изменения ветки branch_2 в ветку branch_1.

```

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1)
$ git merge 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.

```

Рисунок 17. Слияние ветки branch_2 в branch_1

18. Решил конфликт файла 1.txt в ручном режиме

```

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1|MERGING)
$ git add .

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1|MERGING)
$ git status
On branch branch_1
All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:
  modified:   1.txt
  modified:   3.txt

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1|MERGING)
$ git commit -m "merge branch_2 to branch_1"
[branch_1 8ad65cd] merge branch_2 to branch_1

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1)
$

```

Рисунок 18. Решение конфликта вручную с файлом 1.txt

19. Отправил ветку branch_1 на GitHub.

```

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1)
$ git push --set-upstream origin branch_1
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 12 threads
Compressing objects: 100% (18/18), done.
Writing objects: 100% (26/26), 2.16 KiB | 2.17 MiB/s, done.
Total 26 (delta 10), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (10/10), done.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:   https://github.com/Zloybanan4ik/lab3/pull/new/branch_1
remote:
To https://github.com/Zloybanan4ik/lab3.git
 * [new branch]      branch_1 -> branch_1
branch 'branch_1' set up to track 'origin/branch_1'.

```

Рисунок 19. Отправил ветку на GitHub

20. Создал средствами GitHub удаленную ветку branch_3.

```

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_1)
$ git checkout -b branch_3
Switched to a new branch 'branch_3'

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_3)
$ git push origin branch_3
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'branch_3' on GitHub by visiting:
remote:   https://github.com/Zloybanan4ik/lab3/pull/new/branch_3
remote:
To https://github.com/Zloybanan4ik/lab3.git
 * [new branch]      branch_3 -> branch_3

```

Рисунок 20. Создание новой ветки

21. Создал в локальном репозитории ветку отслеживания удаленной ветки branch_3.

```

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_3)
$ git remote add A https://github.com/Zloybanan4ik/lab3.git

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_3)
$ git fetch A
From https://github.com/Zloybanan4ik/lab3
 * [new branch]      branch_1 -> A/branch_1
 * [new branch]      branch_3 -> A/branch_3
 * [new branch]      main -> A/main

```

Рисунок 21. Создание ветки отслеживания изменений удаленной ветки

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

```

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_3)
$ git add .

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_3)
$ git status
On branch branch_3
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   2.txt

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_3)
$ git commit -m "change branch_3"
[branch_3 62edb23] change branch_3
1 file changed, 1 insertion(+)

```

Рисунок 22. Добавление изменений на ветку branch_3

23. Выполнил перемещение ветки master на ветку branch_2.


```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_3)
$ git checkout branch_2
Switched to branch 'branch_2'

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_2)
$ git rebase main
Current branch branch_2 is up to date.
```

Рисунок 23. Перемещение веток

24. Отправил изменения веток master и branch_2 на GitHub.

```
user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_2)
$ git push --set-upstream origin branch_2
gTotal 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'branch_2' on GitHub by visiting
remote:   https://github.com/Zloybanan4ik/lab3/pull/new/branch_2
remote:
To https://github.com/Zloybanan4ik/lab3.git
 * [new branch]      branch_2 -> branch_2
branch 'branch_2' set up to track 'origin/branch_2'.
af

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (branch_2)
$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 6 commits.
  (use "git push" to publish your local commits)

user@DESKTOP-VI4I94J MINGW64 ~/workspace/lab3 (main)
$ git push
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Zloybanan4ik/lab3.git
 41e416b..36bcc70  main -> main
```

Рисунок 24. Внесение изменений в удаленный репозиторий

Выводы: в ходе выполнения лабораторной работы исследовал базовые возможности по работе с локальными и удаленными ветками Git. Научился создавать, а также сливать одну ветку в другую, решать конфликт слияния.

Контрольные вопросы:

1. Что такое ветка? – Ветка в системе контроля версий Git — это отдельная линия разработки проекта, которая позволяет разработчикам работать над разными версиями кода параллельно, не мешая друг другу. Ветви создаются для того, чтобы разработчики могли экспериментировать с новыми функциями, исправлять баги или вносить другие изменения без риска повредить основную версию кода.

2. Что такое HEAD? – HEAD — это специальный указатель, который указывает на текущую рабочую ветку или коммит. Когда вы выполняете команду `git checkout`, HEAD перемещается к выбранной ветке или коммиту. По сути, HEAD показывает, какие файлы и версии файлов находятся в рабочей директории.

3. Способы создания веток. – Есть несколько способов создания новой ветки:

Создать новую ветку и сразу перейти на нее

```
git checkout -b <имя_ветки>
```

Создать новую ветку, но остаться на текущей

```
git branch <имя_ветки>
```

4. Как узнать текущую ветку? – Чтобы узнать, на какой ветке вы находитесь, выполняется следующая команда:

```
git branch
```

Текущая активная ветка будет помечена звездочкой (*).

5. Как переключаться между ветками? – Для перехода на другую ветку используется команда `checkout`. Эта команда перемещает на указанную ветку и обновит содержимое рабочей директории согласно состоянию этой ветки.

6. Что такое удаленная ветка? – Удалённая ветка (remote branch) — это копия ветки, хранящаяся на удалённом репозитории. Она синхронизируется с локальными изменениями при выполнении команд `push`, `fetch` и `pull`.

7. Что такое ветка отслеживания? – Ветка отслеживания — это локальная ветка, которая связана с соответствующей удалённой веткой. При создании такой ветки она автоматически настраивается так, чтобы отслеживать изменения в удаленной ветке. Это удобно для автоматического обновления локальных изменений при выполнении команды `git pull`.

8. Как создать ветку отслеживания? – Когда клонируется репозиторий, обычно уже создаётся одна ветка отслеживания — `master` или `main`. Чтобы создать новую ветку отслеживания, выполняется команда `git checkout -b`.

9. Как отправить изменения из локальной ветки в удаленную ветку? – Для отправки изменений из локальной ветки на удалённый сервер используется команда: `git push`

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

-`git fetch`: Эта команда загружает новые данные из удалённого репозитория, включая изменения в ветках, метаданные и объекты, но не изменяет ваши локальные ветки. Вы можете вручную объединить эти изменения в свою работу после выполнения `fetch`.

- `git pull`: Команда выполняет две операции одновременно: сначала `fetch`, а затем объединяет изменения из удалённой ветки в вашу локальную. То есть она автоматически применяет изменения из удаленного репозитория к вашему рабочему дереву.

11. Как удалить локальную и удаленную ветки? – Удаление локальной и удалённой веток выполняется следующими командами:

- Удалить локальную ветку `git branch -d`

- Если ветка ещё содержит несохранённые изменения, то потребуется использовать параметр `-D` `git branch -D`

- Для удаления удалённой ветки:

`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 присутствуют в модели `git-flow`? Как организована работа светками в модели `git-flow`? В чем недостатки `git-flow`? – модель `git-flow` представляет собой структурированный подход к управлению ветвями в Git. Она ориентирована на проекты с регулярными релизами и часто используется в крупных проектах с несколькими участниками.

Основные типы веток в модели `git-flow`:

1. ****Main (или Master)**** — основная стабильная ветка, содержащая готовый к релизу код. Здесь хранится история релизов.

2. ****Develop**** — главная ветка для разработки новых функций и интеграции изменений. В эту ветку сливаются результаты работы из других веток.

3. ****Feature branches**** — временные ветки, создаваемые для реализации отдельных функциональностей. Они ответвляются от ветки `develop` и возвращаются туда же после завершения работы.

4. ****Release branches**** — ветки, предназначенные для подготовки выпуска новой версии продукта перед релизом. Эти ветки ответвляются от `develop`, и в них вносятся последние правки и доработки перед окончательным релизом.

5. ****Hotfix branches**** — используются для быстрого исправления критических багов в продакшн-версии. Такие ветки ответвляются непосредственно от `main` и могут возвращаться как в `main`, так и в `develop`.

Организация работы с ветками в модели git-flow:

1. Вся разработка начинается с ветки `develop`. Это основной рабочий поток, куда сливаются изменения из веток `feature` и `release`.

2. Новые функциональные возможности разрабатываются в отдельных ветках типа `feature/*`, которые потом сливаются обратно в `develop`

3. Перед выпуском создается ветка `release/*`, где проводится финальное тестирование, доработка документации и прочие подготовительные задачи перед релизом. После завершения релиза изменения объединяются обратно в `master` и в `develop`,

4. Если необходимо исправить ошибку в продакшене, создается ветка типа `hotfix/*`, которая берется с ветки `main`. После исправлений эта ветка возвращается в `master`, а затем изменения сливаются и в ветку `develop`

Недостатки git-flow:

1. ****Сложность для новичков****: Модель требует понимания множества правил и процедур, что делает ее трудной для освоения начинающими разработчиками.

2. ****Избыточность для маленьких проектов****: Для небольших команд или проектов с короткими циклами разработки такая сложная структура может оказаться излишней.

3. ****Много веток****: Поддержание большого количества веток может затруднять управление проектом, особенно когда проект становится крупным и сложным.

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 – Одним из популярных программных средств с графическим интерфейсом для работы с Git является SourceTree. Этот инструмент предоставляет удобный набор функций для работы с ветками Git. Рассмотрим основные инструменты SourceTree для работы с ветками:

Создание ветки

Я могу легко создать новую ветку в SourceTree следующим образом:

1. Переходишь в раздел "Branches".
2. Нажимаешь на кнопку "New Branch".
3. Указываешь имя новой ветки и выбираешь точку, от которой она будет создана (чаще всего это текущая ветка или последний коммит).

Переключение между ветками

Для переключения между ветками в SourceTree я делаю следующее:

1. Открываю раздел "Branches".
2. Нахожу нужную ветку в списке и дважды щелкаю на ней.

Просмотр истории веток SourceTree имеет встроенный график коммитов, позволяющий мне просматривать историю веток. Для этого:

1. Я перехожу в раздел "History".
2. Выбираю интересующую меня ветку, и SourceTree показывает все коммиты, сделанные в этой ветке.

Объединение веток (Merge) Для слияния одной ветки с другой в SourceTree:

1. Убеждаюсь, что нахожусь на той ветке, в которую хочу слить изменения.

2. Переходя в раздел "Actions", выбираю "Merge...".

3. Выбираю ветку, изменения из которой хочу слить, и нажимаю "ОК".

Отправка изменений в удалённый репозиторий

Для отправки изменений на удалённый сервер:

1. Делаем необходимые коммиты в моей локальной ветке.

2. В разделе "Repository" нажимаю на кнопку "Push".

3. В появившемся окне выбираю ветку для отправки и нажимаю "ОК".

Удаление веток

Для удаления ветки в SourceTree нужно сделать следующее:

1. Найти ветку, которую хочешь удалить, в разделе "Branches".

2. Щёлкнуть правой кнопкой мыши на ветке и выбрать "Delete Branch".

Работа с удалёнными ветками SourceTree также поддерживает работу с удалёнными ветками. Для взаимодействия с ними:

1. В разделе "Remote Repositories" выбираю нужный удалённый репозиторий.

2. Использую соответствующие кнопки для создания, удаления, фетчинга и пушинга веток.

Таким образом, SourceTree действительно удобен для работы с ветвями Git, делая этот процесс более наглядным и простым.

Ссылка на репозиторий: <https://github.com/Zloybanan4ik/lab3.git>