**Лабораторная работа 1 Hello GitHub и средство контроля версий Git**

Git — это система управления версиями с открытым исходным кодом. Она упрощает совместную работу над проектами с помощью *распределенной системы управления версиями* файлов (бывают ещё локальные и централизованные), которые хранятся в *репозиториях*.

GitHub — это служба размещения в Интернете репозиториев Git. Некая платформа для размещения кода для контроля версий и совместной работы над одним проектов из любого места. С помощью GitHub можно создавать, распространять и поддерживать программное обеспечение.

**Цель:**
познакомиться с одним из средств контроля версий через веб-сервис GitHub.

**Задачи:**

1. Завести аккаунт на GitHub (если еще нет)
2. Познакомиться с основными возможностями GitHub
   2.1 Создать репозиторий
   2.2 Создать ветку
   2.3 Внести и сохранить изменения
   2.4 Слить изменения
3. Использование консоли для Git
   3.1 Установка Git
   3.2 Настройка Git
   3.3 Клонировать репозиторий с GitHub на компьютер
   3.4 Создание, изменение и слияние ветки
   3.5 Отмена изменений

#### 1 Создание аккаунта

Тут вопросов возникнуть не должно. Заходим на https://github.com/ и нажимаем “Sign up”, понадобится адрес электронной почты, придумать пароль и имя пользователя.

### 2 Основные возможности

### 2.1 Создание репозитория

Как правило в одном репозитории располагается один проект. Он может содержать папки и файлы, любые файлы, не только код, всё, что нужно для проекта.

Обычно в репозитории располагается README файл написанный с помощью синтаксиса **Markdown** (упрощенный язык разметки). В README содержится информация о проекте.

Для создания репозитория в верхнем правом углу на странице вашего аккаунта нужно нажать кнопку “**+**” и в всплывающем меню выбрать пункт “**New repository**” (“Новый репозиторий”).

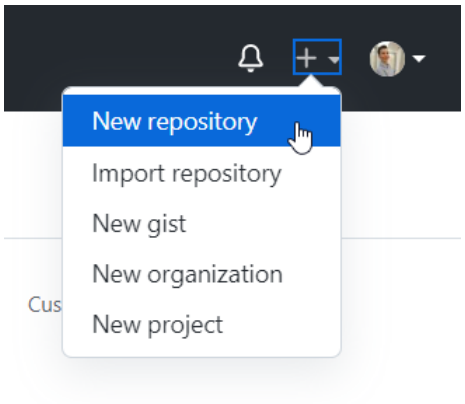



Рисунок 1. Создание репозитория

Нужно придумать имя репозитория, назовём наш учебный репозиторий “Information System Tools.

Заполним поля:

**Repository name = Information-Systems-Tools**
**Description = <напишите короткое описание проекта>**

Нужно выбрать, что проект будет публичный, отмечаем “**Public**”.
Выбираем добавить файл README, “**Add a README file**”.
И нажимаем кнопку “**Create repository**” (“Создать репозиторий”).

Репозиторий готов, переходим к его наполнению.

### 2.2 Создание ветки

Создание ветки (branch) позволяет иметь различные версии репозитория в одно и то же время. Изначально в проекте одна ветка, которая называется main. Если требуется добавить какую-то функцию и при этом нельзя менять основной проект, то можно создать дополнительную ветку проекта для отладки этой функции. Можно производить различные изменения в дополнительной ветке и это никак не будет влиять на содержимое и работу главной ветки, пока мы не сольем (merge) дополнительную ветку с главной.

При создании ответвления main в новую ветку копируется текущее состояние main. Если кто-то другой обновил main, пока вы работали в своей ветке, то можно будет перенести (pull) эти обновления в вашу ветку.

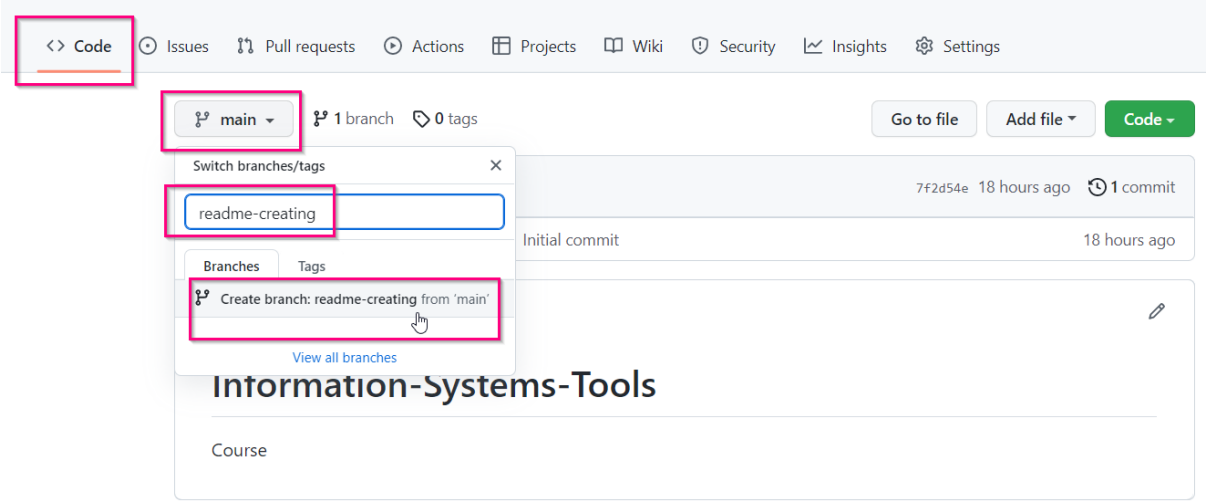



Рисунок 2. Создание ветки

Непосредственное создание новой ветки (смотрите рисунок 2):

1. Переходим на вкладку **Code.**
2. Открываем выпадающее меню или панель справа от **main**
3. Вводим имя новой ветки в поле “**Find or create a branch…**”, пусть будет “**readme-creating**”.
4. Нажимаем “**Create branch: <имя вашей ветки> from “main”**”.

Теперь имеем две ветки и они на данный момент идентичны.

### 2.3 Внести и сохранить изменения

Добавим изменение в нашу новую ветку **readme-creating**.

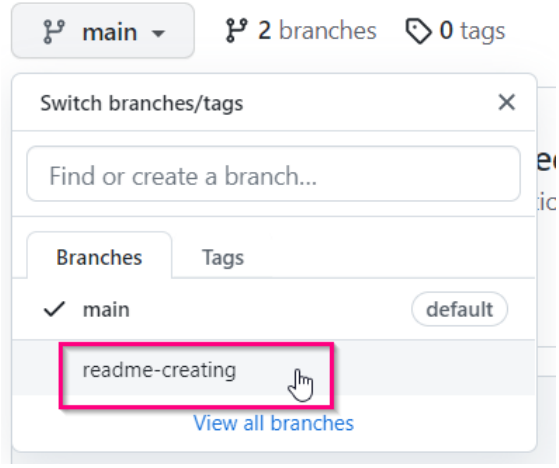



Рисунок 3 Переключение между ветками

Если выполняете работу последовательно, то на данном этапе новая ветка будет активна. Иначе необходимо перейти в новую ветку, это можно сделать через ниспадающую панель справа от **main**, как показано на рисунке 3.

В Git сохранение изменений или фиксация изменений называется так же как в базах данных – **commit**. Далее будем использовать этот термин.

Каждый коммит должен сопровождаться пояснением, почему было внесено данное изменение. Пишите подробно и по пунктам, что изменилось, чтобы другим и вам в будущем было понятно зачем были сделаны изменения. (Данная рекомендация касается не только Git).

Приступим:

1. Открываем **readme** файл, кликаем по нему.
2. Файл **readme** должен выглядеть как на рисунке 4, и чтобы начать его редактировать кликаем “**Edit this file**”.

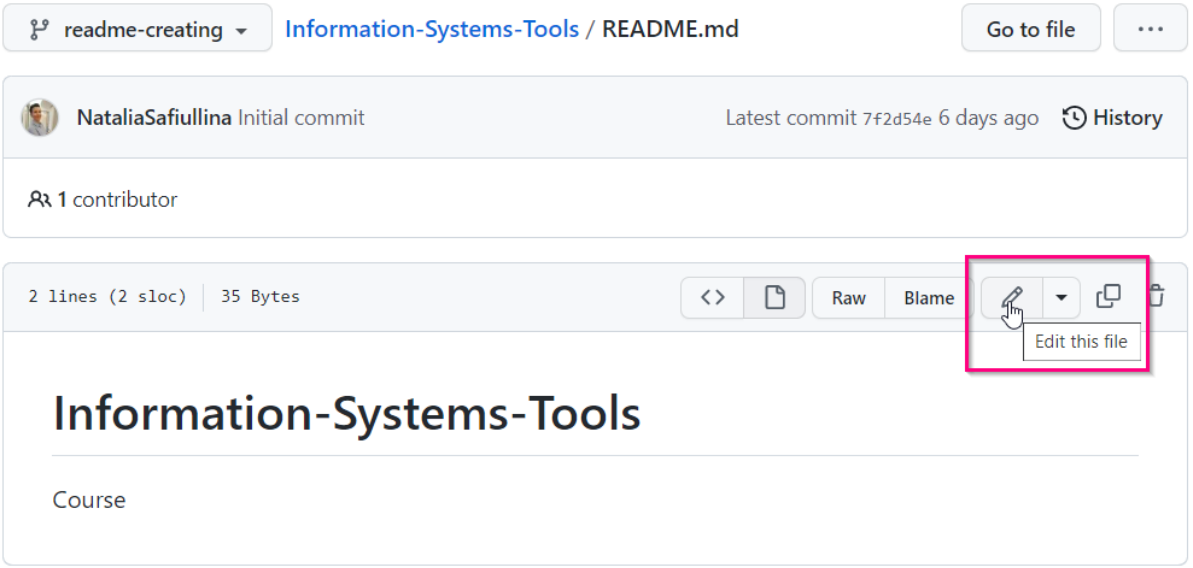



Рисунок 4 Запуск редактирования файла

3. Напишите немного о себе используя как можно больше синтаксис **Markdown**. Ниже немного примеров синтаксиса:

| Формат | Синтаксис |
|---|---|
| Заголовки различных уровней | `# H1`<br>`## H2`<br>`### H3` |
| Жирный шрифт | `**bold text**` |
| Курсив | `*italicized text*` |
| Цитата | `> …` |

| | |
|---|---|
| Программный код | ‘...’ |
| Ссылка | [текст который будет ссылкой](https://www.example.com) |
| Картинка | ![подпись](image.jpg) |
| Нумерованный список | 1. …<br>2. …<br>3. … |
| Ненумерованный список | – …<br>– …<br>– … |

Поищите в интернете остальные элементы синтаксиса, если они вам нужны.
Просмотреть предварительный результат редактирования можно перейдя во вкладку **Preview** (рисунок 5).

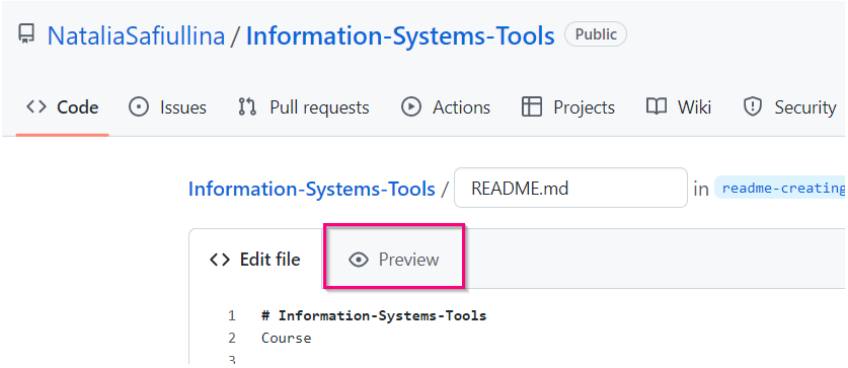



Рисунок 5 Предварительный просмотр README файла

4. Ниже поля редактирования находится область **Commit changes** (рисунок 6). Перед сохранением изменений добавьте описание изменений.

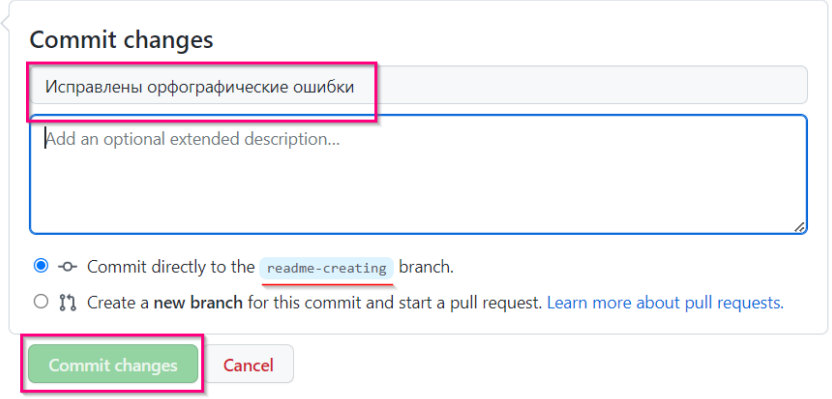



Рисунок 6 Область Commit changes

5. Сохраним изменения, кликаем кнопку “**Commit changes**”, изменения сохранятся в ветке **readme-creating**, теперь она отличается от **main**.

### 2.4 Слить изменения

Если изменения готовы и их можно слить с основной веткой проекта, то начинается **pull request**, т.е. запрос на изменение кода. Он показывает различия в содержании обоих веток.

Обычно слияние происходит при командной работе, т.е. изменения предлагаются на рассмотрение другим людям, но можно предложить и себе для эксперимента и обучения.

1. Переходим на вкладку **Pull Requests** (рисунок 7).
2. Нажимаем кнопку **New pull request**.

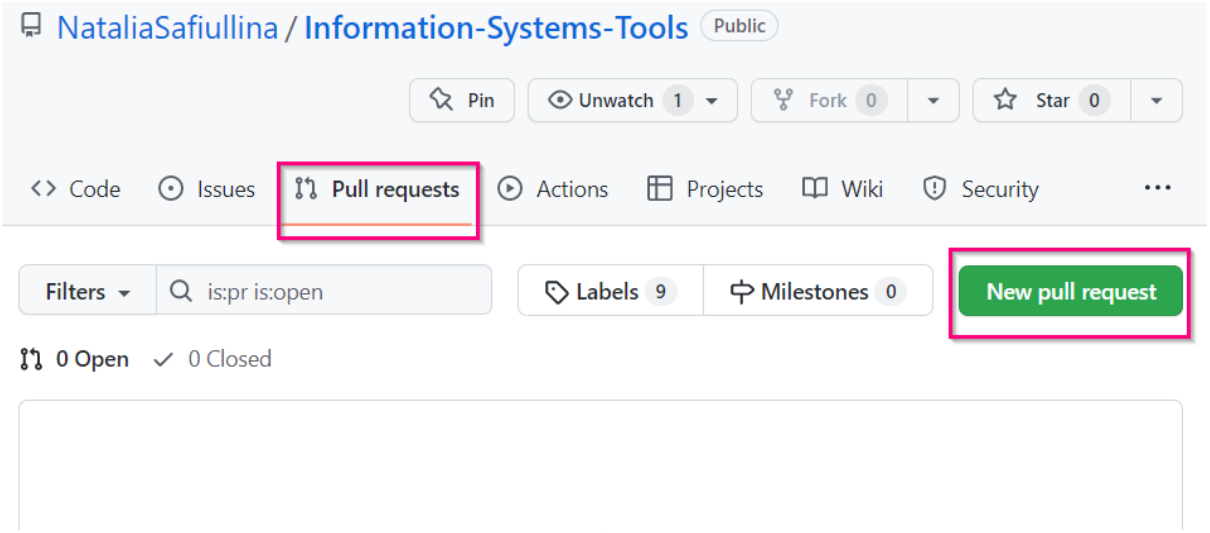



Рисунок 7 Создание pull request

3. В области **Example Comparisons** (рисунок 8) выбираем ветку **readme-creating** и в качестве ветки, с корой сравниваем, указана **main**.

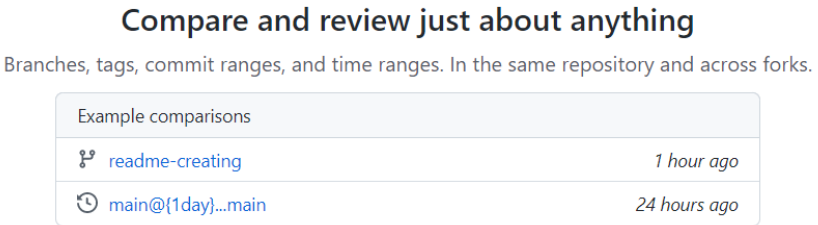



Рисунок 8 Выбор ветки для слияния

4. На следующем шаге видим что мы хотим предложить, проверяем не ошиблись ли мы с выбором изменений. Git информирует какие изменения есть, добавления, удаления и прочее.
5. Теперь создаём запрос на изменения, нажимаем **Create pull request**, необходимо придумать название и краткое описание (рисунок 9).

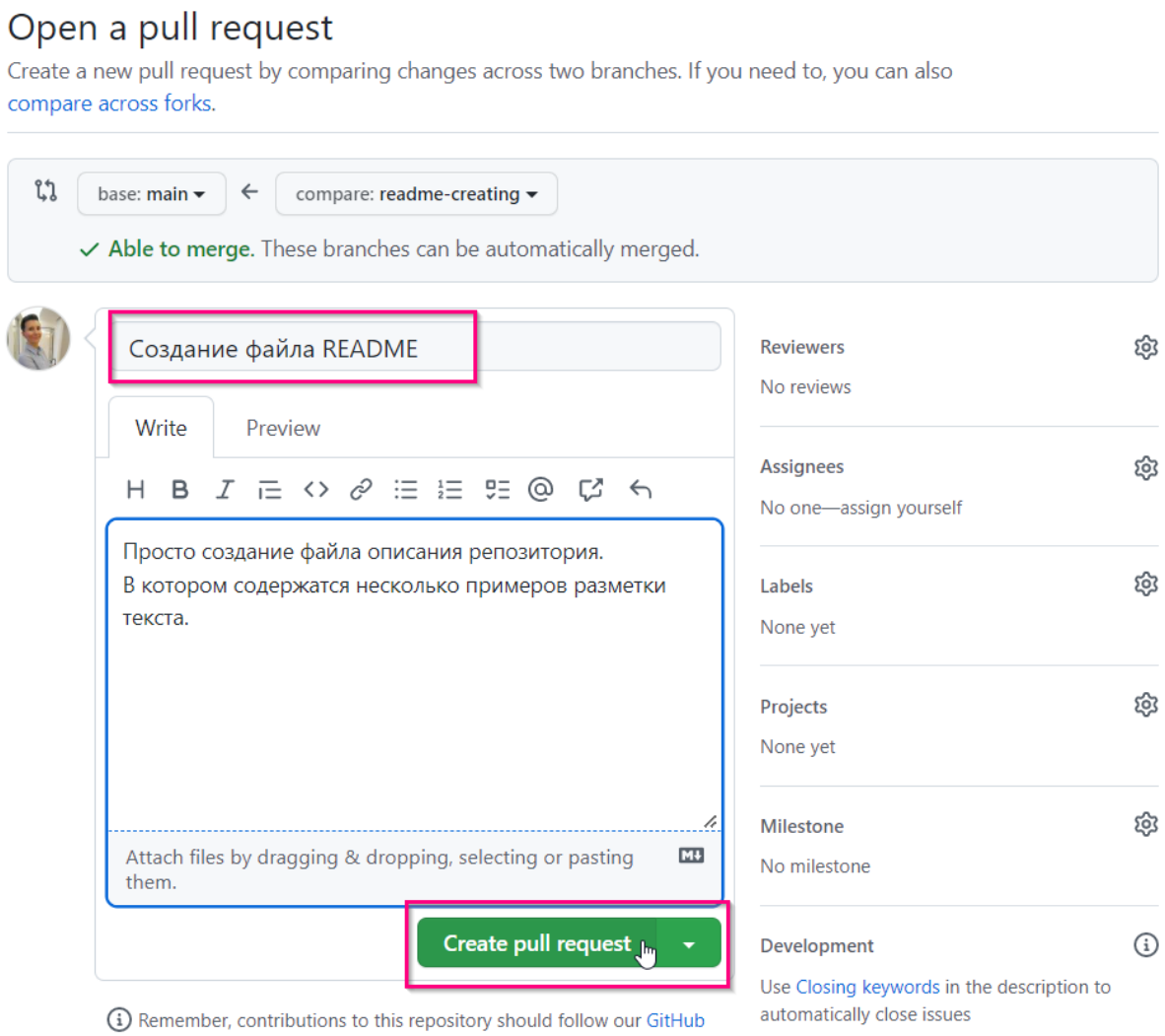



Рисунок 9 Описание изменений

6. Кликаем **Create pull request**

Соавторы могут проверить, что вы предлагаете и вынести свои предложения. Переходим к последнему этапу непосредственному объединению веток.

Если в двух версиях будут конфликты, то GitHub предупредит об этом и не даст объединить ветки, пока конфликт не будет разрешён.

В наших изменениях не должно быть никаких конфликтов, поэтому:

1. Нажимаем **Merge pull request**.

2. Нажимаем **Confirm merge**.
3. Нажимаем **Delete branch**, чтобы удалить ветку **readme-creating**.

## 3 Использование консоли для Git

Существует консольное приложение и для Git  и для GitHub. Консольное приложение для Git называется тоже Git (что не удивительно) и его можно скачать тут: https://git-scm.com/, там вы найдёте установочный файл, который кроме консольного приложения содержит также Git GUI (т.е. графическую оболочку Git (Graphical User Interface)), Git Bash (эмулирует работу командной строки Linux/unix), Git CMD (командная строка windows). Консольная утилита для GitHub называется GitHub CLI, или gh.

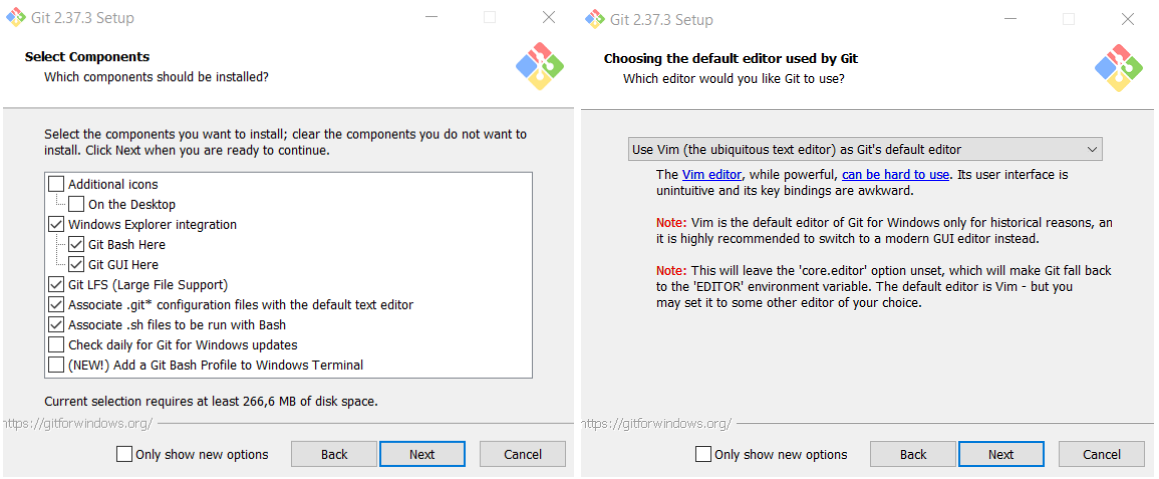
Консольные утилиты и консольные приложения нужны чтобы можно было воспользоваться ими внутри программы или скрипта. Поэтому рассмотрим только консольные утилиты и пропустим GUI.

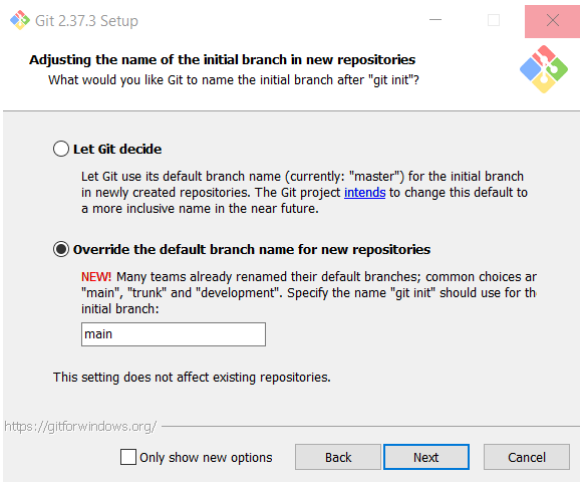
Рассмотрим основные возможности консольной утилиты Git.

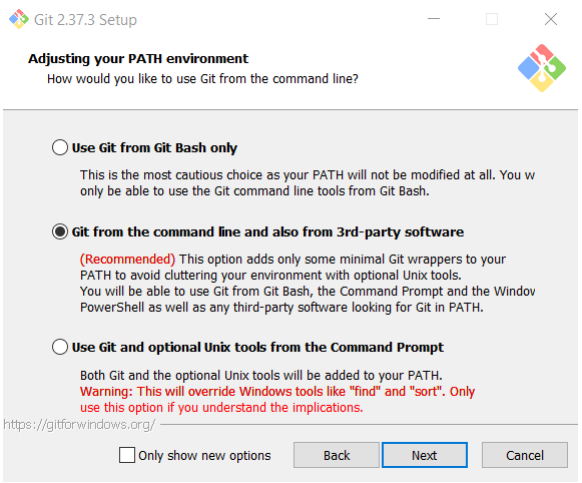
### 3.1 Установка Git

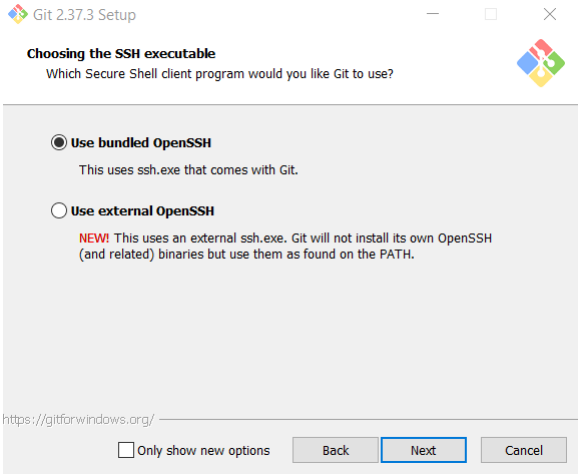
Скачайте Git подходящий для вашей ОС и установите его. Для Windows скачивайте тут: https://git-scm.com/download/win .

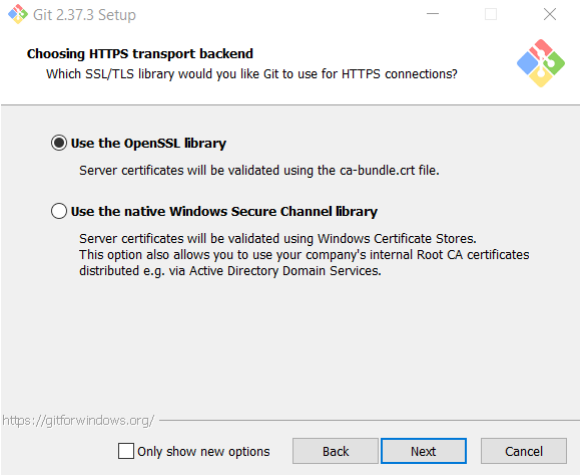
Следуйте инструкции установки.

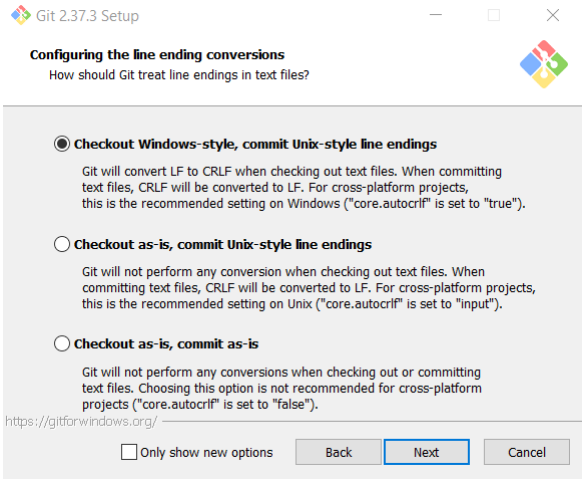


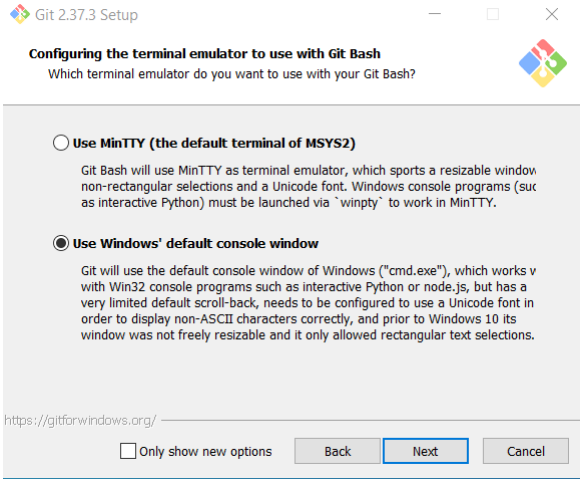

Git 2.37.3 Setup
Adjusting the name of the initial branch in new repositories
What would you like Git to name the initial branch after "git init"?
Let Git decide
Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project intends to change this default to a more inclusive name in the near future.
Override the default branch name for new repositories
NEW! Many teams already renamed their default branches; common choices ar "main", "trunk" and "development". Specify the name "git init" should use for th initial branch:
main
This setting does not affect existing repositories.
https://gitforwindows.org/
Only show new options
Back
Next
Cancel

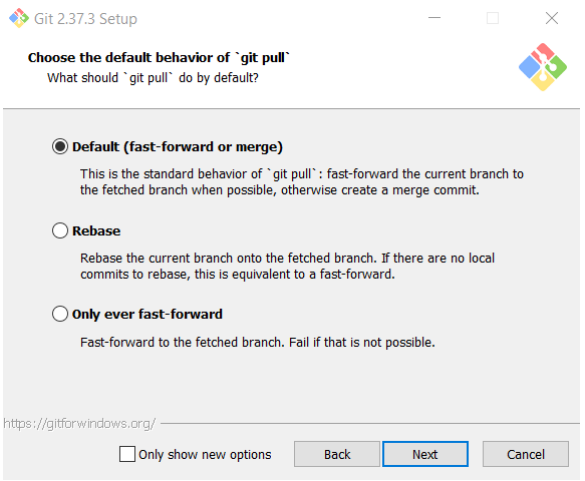

Git 2.37.3 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the most cautious choice as your PATH will not be modified at all. You w only be able to use the Git command line tools from Git Bash.
Git from the command line and also from 3rd-party software
(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools.
You will be able to use Git from Git Bash, the Command Prompt and the Windov PowerShell as well as any third-party software looking for Git in PATH.
Use Git and optional Unix tools from the Command Prompt
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.
https://gitforwindows.org/
Only show new options
Back
Next
Cancel

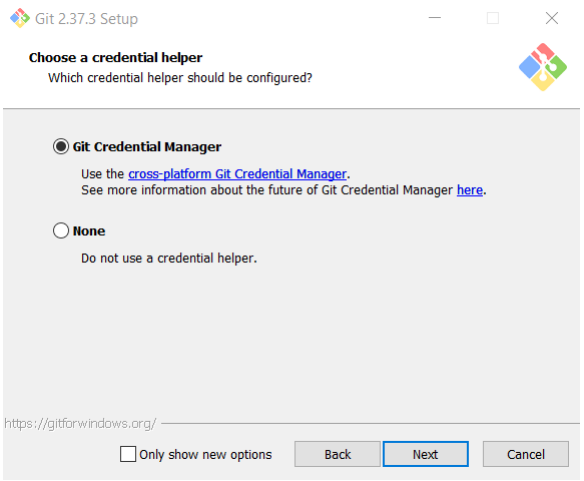

Git 2.37.3 Setup
Choosing the SSH executable
Which Secure Shell client program would you like Git to use?
Use bundled OpenSSH
This uses ssh.exe that comes with Git.
Use external OpenSSH
NEW! This uses an external ssh.exe. Git will not install its own OpenSSH (and related) binaries but use them as found on the PATH.
https://gitforwindows.org/
Only show new options
Back
Next
Cancel

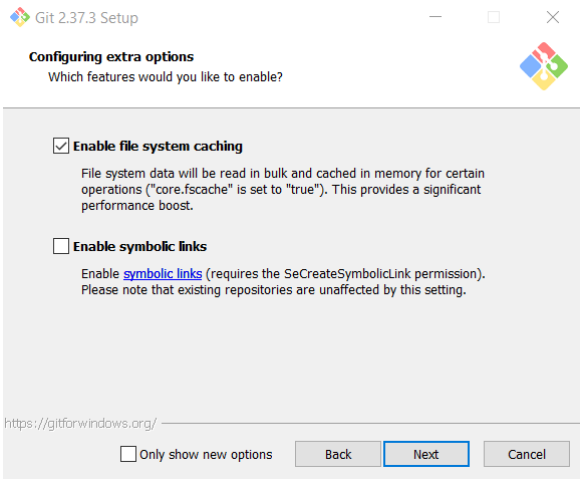

Git 2.37.3 Setup
Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?
Use the OpenSSL library
Server certificates will be validated using the ca-bundle.crt file.
Use the native Windows Secure Channel library
Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.
https://gitforwindows.org/
Only show new options
Back
Next
Cancel

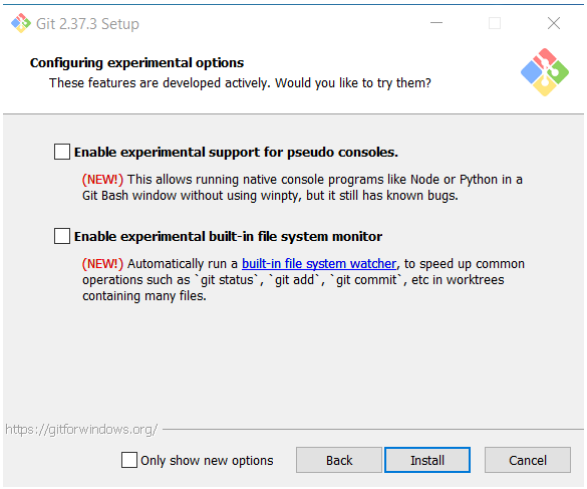

Git 2.37.3 Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").
https://gitforwindows.org/
Only show new options
Back
Next
Cancel


Git 2.37.3 Setup
Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?
Use MinTTY (the default terminal of MSYS2)
Git Bash will use MinTTY as terminal emulator, which sports a resizable windov non-rectangular selections and a Unicode font. Windows console programs (suc as interactive Python) must be launched via `winpty` to work in MinTTY.
Use Windows' default console window
Git will use the default console window of Windows ("cmd.exe"), which works v with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.
https://gitforwindows.org/
Only show new options
Back
Next
Cancel




Проверьте, что Git установлен:

1. запустите командную строку,
2. введите команду: **`git version`**.

Должен быть получен ответ, например:

```
d:\GIT\Information-Systems-Tools>git version
git version 2.37.3.windows.1
```

### 3.2 Настройка Git

Подумайте, где вы хотите разместить репозитории Git. Перейдите туда, создайте нужные папки. Например, пусть Git располагается на диске D:\ в папке “GIT”, так что переходим на диск D:\ и создаем папку:

```
C:\>d:
d:\>md GIT
d:\>cd GIT
d:\GIT>
```

У нас уже есть учётная запись и репозиторий на GitHub, подключим Git CMD к нашей учётной записи.

Вводим две команды:

```
git config --global user.name "MyName"
git config --global user.email "MyEmail@example.com"
```

### 3.3 Клонировать репозиторий с GitHub на компьютер

Попробуем скопировать (клонировать) репозиторий Information-System-Tools, который мы создали с помощью GitHub на компьютер. Для этого:

1. Получаем URL репозитория, который можно скопировать в ниспадающей панели **Code** (рисунок 12).

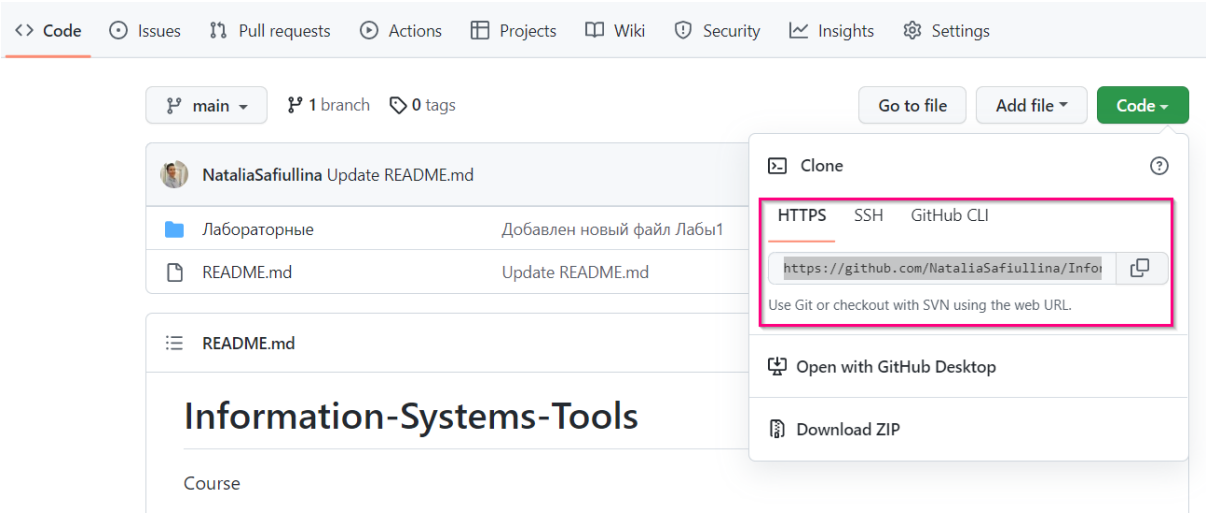



Рисунок 12 Получить URL репозитория

2. Выполняем команду для клонирования репозитория:

```
git clone https://github.com/<аккаунт>/<репозиторий>.git
```

В итоге Git сделает клонирование репозитория в папку с названием репозитория (папку создаст сам) и сообщит, когда закончит:

```
Cloning into 'Information-Systems-Tools'...
remote: Enumerating objects: 48, done.
```

```
remote: Counting objects: 100% (48/48), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 48 (delta 10), reused 0 (delta 0), pack-reused 0

Receiving objects: 100% (48/48), 452.41 KiB | 661.00 KiB/s, done.
Resolving deltas: 100% (10/10), done.
```

Внутри папки репозитория Git создаст скрытую папку “.git” (в ней содержится всё необходимое для работы Git) и начнёт отслеживать изменения.

3. Переходим внутрь папки репозитория, команда **cd**.

```
cd Information-Systems-Tools
```

Проверьте что получилось, посмотрите файлы и папки репозитория, например через проводник Windows.

Выполните команду status:

```
git status
```

– эта команда показывает состояния файлов в рабочем каталоге: какие файлы изменены, но не добавлены; какие ожидают коммита; кроме этого она показывает текущую ветку:

```
d:\GIT\Information-Systems-Tools>git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Выполните команду log:

```
git log
```

– перечисляет коммиты, сделанные в репозитории в обратном к хронологическому порядке.

Пролистайте коммиты стрелкой вниз, пока не появится слово (END), затем нажмите “q” на клавиатуре, чтобы выйти из истории коммитов. (У команды log есть несколько полезных аргументов и опций).

### 3.4 Создание, изменение и слияние ветки

Создать новую ветку можно двумя способами, введите любой из этих вариантов:

вариант 1

```
git branch new-branch
git checkout new-branch
```

или вариант 2

```
git checkout -b new-branch
```

Команды:

**git branch [имя ветки]**– создание новой ветки, без параметров показывает все существующие ветки и отмечает активную.

**git checkout [имя ветки]** – переключение в указанную ветку, опция -b означает, что ветку надо сначала создать и потом переключиться, у checkout много других опций и действий.

Теперь мы работаем в новой ветке. Выполните опять команду:

```
git status
```

Git должен написать что мы находимся в новой ветке и сообщит, что нечего коммитить. Создайте внутри папки репозитория новую папку и внутри неё новый файл. И снова выполните:

```
git status
```

Теперь Git перечислит не отслеживаемые файлы и порекомендует добавить их в коммит, последней строкой сообщит, что ничего не добавлено в коммит, но есть неотслеживаемые файлы:

```
d:\GIT\Information-Systems-Tools>git status
On branch new-branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        TEST/

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

Далее добавляем нашу новую папку и файл в коммит (или по другому в индекс, т.е. в список того, что будет коммититься), выполняем команду:

**`git add .`**

Команда:
**`git add [имя папки | имя файла | .]`** – добавляет измененные файлы в индекс, “.” добавляет все файлы.

Снова выполняем команду:

**`git status`**

Теперь Git сообщает, что есть изменения которые будут коммититься:

```
d:\GIT\Information-Systems-Tools>git add .

d:\GIT\Information-Systems-Tools>git status
On branch new-branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   TEST/test.txt
```

Давайте сохраним наши изменения, выполним команду:

**`git commit -m "Add a new folder and a new file"`**

Команда:
**`git commit [-m “комментарий”]`** – сохраняет измененные файлы в репозитории, **`-m`** тоже самое, но с комментарием.

*Если не указать -m, то попадем в выбранный при установке текстовый редактор, у нас это был Vim. В нем пишем комментарий в первой строке, чтобы выйти из него нужно нажать Esc, ввести “:”, ввести “x” (это команда сохранить и выйти), нажать Enter.*

Проверим на GitHub, изменился ли наш репозиторий. Никаких изменений быть не должно. До этого момента мы меняли локальный репозиторий. Для внесения изменений в удаленный репозиторий выполним команду **`git push`**, специальную разновидность для ветки:

**`git push --set-upstream origin`** *`new-branch`*

Команда:
**`git push`** – передает коммиты из локального репозитория в удаленный.

Должны получить:

```
d:\GIT\Information-Systems-Tools>git push --set-upstream origin new-branch
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 420 bytes | 210.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'new-branch' on GitHub by visiting:
remote:
https://github.com/NataliaSafiullina/Information-Systems-Tools/pull/new/new-bran
ch
remote:
To https://github.com/NataliaSafiullina/Information-Systems-Tools.git
 * [new branch]      new-branch -> new-branch
branch 'new-branch' set up to track 'origin/new-branch'.
```

Если вы комитете изменения в удаленный репозиторий первый раз, то возможно потребуется авторизация (рисунок 13).

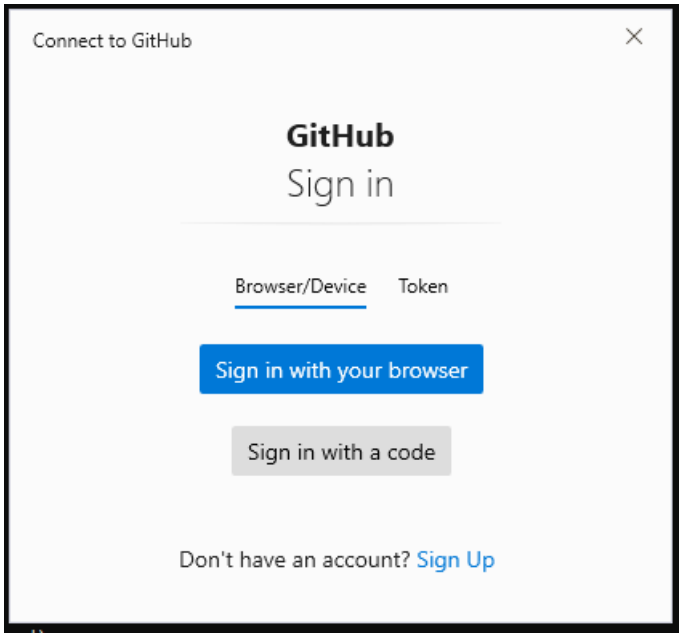



Рисунок 13 Авторизация

Нужно авторизоваться по токену (PAT, personal access token). Чтобы его создать, выполняем следующие шаги:

1. заходим в GitHub в Settings (рисунок 14),
2. далее находим Developer settings,
3. далее заходим в Personal access tokens
4. нажимаем “Generate new token”,

5. заполняем параметры (рисунок 15), для работы с командной строкой выбираем repo.

Создаем токен и копируем его себе, так как больше мы его не увидим. В форме на рисунке 13 переходим во вкладку Token, вводим свой токен.

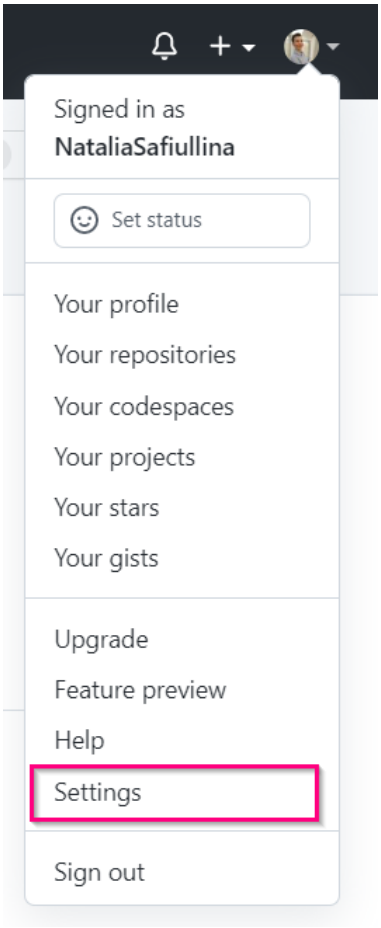



Рисунок 14 Настройки аккаунта

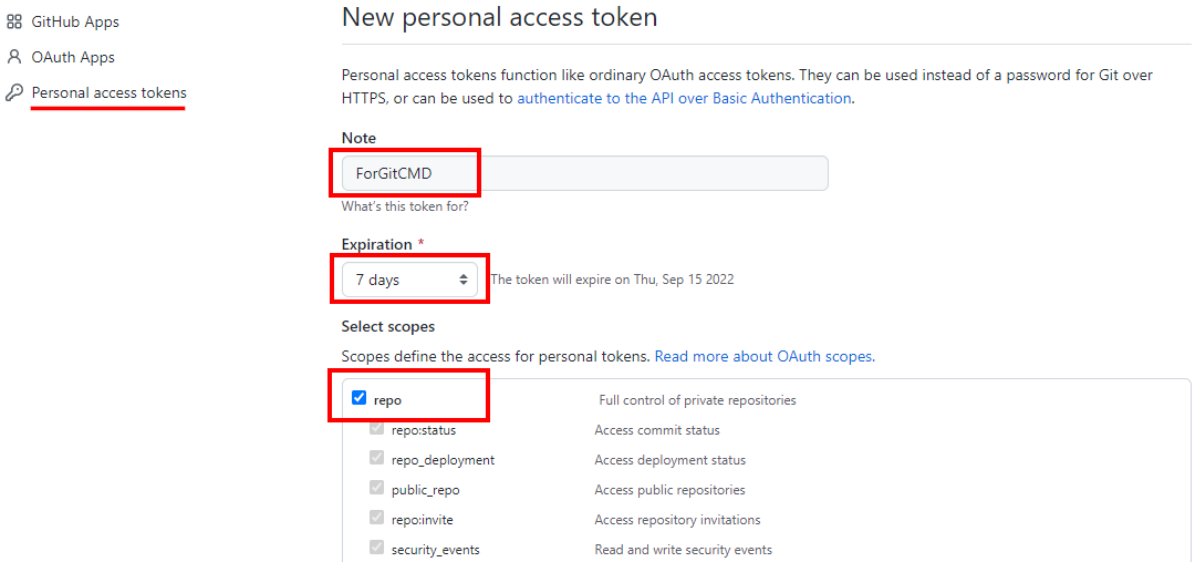



Рисунок 15 Заполнение параметров токена

После того как команда push выполнится в репозитории на GitHub появится наша новая ветка со всеми файлами с локального компьютера.

Выполним слияние. Чтобы слить новую ветку с веткой main, сначала нужно перейти в ветку main, выполняем команду:

```
git checkout main
```

Команда слияния:

```
git merge new-branch
```

Получим:

```
d:\GIT\Information-Systems-Tools>git merge new-branch
Updating 258a484..35d3cf3
Fast-forward
 TEST/test.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 TEST/test.txt
```

Если проверить на GitHub, то там ничего не поменялось, мы также видим две ветки. Нужно опять передать коммиты, выполняем команду:

```
git push
```

Теперь на GitHub изменения появились. И мы видим папки и файлы из ветки new-branch в ветке main.

### 3.5 Отмена изменений

Откатим наш репозиторий на шаг назад, например, удалим последний коммит. Есть несколько способов, мы воспользуемся командой revert и идентификатором коммита.

Сначала получим идентификатор последнего коммита, выполним команду log:

```
git log
```

```
d:\GIT\Information-Systems-Tools>git log
commit 27fffc31032c3b4beebbf0527aa2a681019a5bd7 (HEAD -> main,
origin/new-branch, origin/main, origin/HEAD, new-branch)
```

```
Author: NataliaSafiullina <nfsafiullina@gmail.com>
Date:   Thu Sep 8 22:29:52 2022 +0700

    Add a new folder and a new file
```

Копируем идентификатор выделенный желтым шрифтом и вставляем его в следующую команду:

```
git revert 27fffc31032c3b4beebbf0527aa2a681019a5bd7
```

Проверьте данные на GitHub, ничего не поменялось. Нужно опять отправить изменения в удаленный репозиторий (в данном случае новый коммит удаляет указанный коммит), выполняем:

```
git push
```

Перезайдите в репозиторий на GitHub, изменения пропали.

Конечно у Git существует ещё много команд и различных опций, остановимся пока на этом.

### 4 Контрольные вопросы

1. Для чего нужны инструменты контроля версий?
2. Что такое Git, что такое GitHub? Чем отличаются?
3. Что такое pull request?
4. Какие действия мы проделали через консоль Git?

```
Приложение. Краткий список команд консоли:
git log -- история коммитов.
git status -- измененные файлы (показывает добавлены в коммит или нет).
git add file -- добавить файл в коммит.
git add . -- добавить все изменённые файлы в коммит.
git commit -m 'text' -- добавить подпись коммитов.
git commit --amend -- изменения сообщение последнего коммита.
git branch -- посмотреть ветки.
git branch -v -- просмотре веток с последним в ней коммитом.
git branch -d название ветки -- удалить ветку.
git checkout название ветки -- переключиться в ветку.
git checkout -b название ветки -- создать новую ветку и сразу в неё переключиться.
git push сервер ветка -- залить изменения на сервер в указанную ветку.
git push -f  -- залить изменения на сервер в режиме force, то есть с
возможностью переписать уже имеющиеся коммиты на сервере. Будьте очень
аккуратны с этой командой, а лучше минимизируйте её использование, ведь вы
будете переписывать серверные файлы.
```