**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА   
И ГОСУДАРСТВЕННОЙ СЛУЖБЫ**

**ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**НИЖЕГОРОДСКИЙ ИНСТИТУТ УПРАВЛЕНИЯ – филиал РАНХиГС**

Факультет: Управления

Кафедра Информатики и информационных технологий

Направление подготовки / специальность: 09.03.03 Прикладная информатика

*(код, наименование)*

Направленность (профиль) / специализация: Корпоративные информационные системы управления

**ЛАБОРАТОРНАЯ РАБОТА №1**

|  |  |
| --- | --- |
| по дисциплине: | Программная инженерия |
| на тему: | «Работа с системой контроля версий GIT» |
|  |  |
|  |  |

|  |  |  |
| --- | --- | --- |
|  |  | АВТОР |
|  |  | Обучающийся(иеся) 3 курса группы Иб-531 |
|  |  | заочной формы обучения |
|  |  |  |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_\_ Лазарев С.А. |
|  |  | *(подпись) (фамилия, инициалы)* |
|  |  |  |
|  |  | РУКОВОДИТЕЛЬ |
|  |  | Кандидат физико-математических наук, Доцент |
|  |  | *(ученая степень, ученое звание)* |
|  |  | оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  | «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_ г. |
|  |  | *(дата защиты)* |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_\_ Окулич В.И. |
|  |  | *(подпись) (фамилия, инициалы)* |

Нижний Новгород, 2023 г.

ОГЛАВЛЕНИЕ

[1 Обзор репозитория GitHub 2](#_Toc125543284)

[1.1 Основные термины 2](#_Toc125543285)

[1.2 Работа с GitHub 3](#_Toc125543286)

[1.3 Основные команды GIT 6](#_Toc125543287)

[1.3.1 Создание репозитория 7](#_Toc125543288)

[1.3.2 Клонирование репозитория 7](#_Toc125543289)

[1.3.3 Добавления файлов для коммита 7](#_Toc125543290)

[1.3.4 Проверка состояния файлов в рабочем каталоге 7](#_Toc125543291)

[1.3.5 Просмотр изменений 8](#_Toc125543292)

[1.3.6 Сохранение изменений 8](#_Toc125543293)

[1.3.7 Отмена изменений 8](#_Toc125543294)

[1.3.8 Удаление файлов из индекса 8](#_Toc125543295)

[1.3.9 Перемещение файла 8](#_Toc125543296)

[1.3.10 Менеджер веток 9](#_Toc125543297)

[1.3.11 Переключение веток 9](#_Toc125543298)

[1.3.12 Слияние веток 9](#_Toc125543299)

[1.3.13 Просмотр истории коммитов 9](#_Toc125543300)

[1.3.14 Сбор изменений из удалённого репозитория 9](#_Toc125543301)

[1.3.15 Слияния изменений из удалённого репозитория с локальным репозиторием 9](#_Toc125543302)

[1.3.16 Отправление локальный изменений в удалённый репозиторий 10](#_Toc125543303)

# 1 Обзор репозитория GitHub

**GitHub** — это облачная платформа для хостинга IT-проектов и совместной разработки, под капотом которой находится популярная система контроля версий Git, а также полноценная социальная сеть для разработчиков.

Здесь можно найти множество open-source проектов на разных языках и поучаствовать в них, разместить своё портфолио с примерами кода, чтобы приложить ссылку к резюме, подглядывать в открытых проектах интересные архитектурные решения, смотреть, как опытные разработчики пишут код, и скачивать огромное количество полезных в разработке и бесплатных инструментов для разработки. Некоторые авторы способны собирать в GitHub целые библиотеки — книг и статей, которые можно использовать при изучении новой технологии или языка программирования, и даже для обучения новой специализации в сфере IT.

## 1.1 Основные термины

**Репозиторий** - это корневая папка с файлами и вложенными директориями вашей программы — и одновременно её страница на GitHub. Загрузить в репозиторий можно всё что угодно, но предполагается, что вы будете хранить в нём файлы с исходным кодом и какие-нибудь дополнительные материалы — допустим, необходимую для GUI или вёрстки графику (картинки, иконки и тому подобное).

Репозитории могут быть публичными и приватными, в них можно создавать другие папки и отслеживать изменения версий. Управлять своими репозиториями можно прямо через интерфейс сайта, командную строку, десктопное приложение GitHub или различные средства разработки (IDE), поддерживающие интеграцию с сервисом.

**Ветка (branch)** - В ветки группируются изменения и обновления — допустим, одна главная ветка (по умолчанию создаётся main) и одна beta. Ветки независимы друг от друга, но при желании их можно объединять (merge — слияние) — даже если между ними есть разница в коде.

**Коммит (commit)** – Внесение изменения в содержимое репозитория напрямую или путём создания копии. Данное действие производиться локально на той машине, на которой осуществляется работа.

**Пуш (push)** – перенос локальных изменений, который были закомментированы командой commit, из локального репозитория на удалённый (например: на GitHub)

**Клонирование (clone)** – Копирования данных с удалённого репозитория на локальный компьютер для работы с ними.

**Форк (fork)** – создание отдельной копии репозитория для ведения разработки по-своему. Например, когда хочется развивать какой-либо проект независимо от существующей разработки.

## 1.2 Работа с GitHub

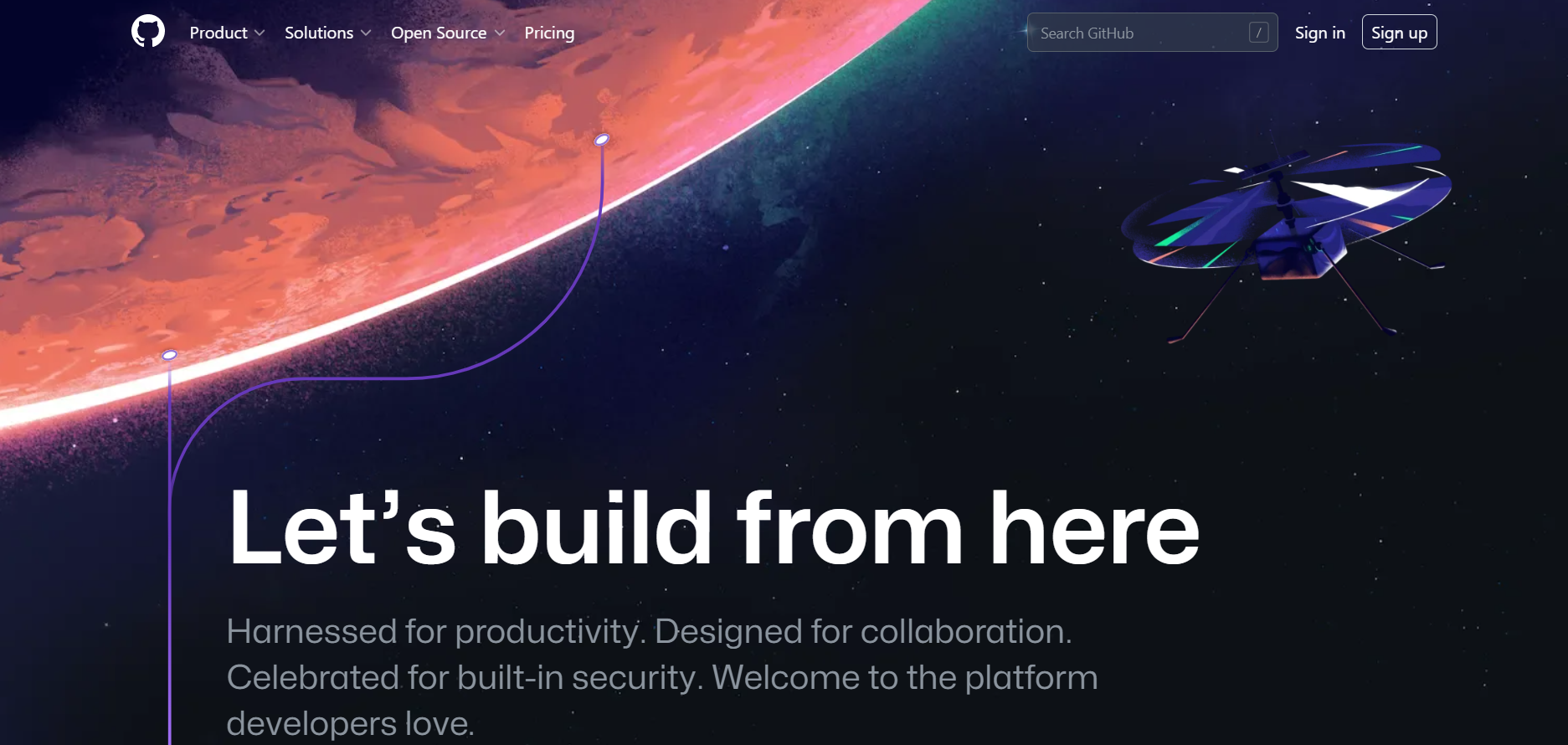


Рис. Фрагмент страницы приветствия GitHub

На рисунке 1 представлена страница приветствия GitHub. Для работы с ним, необходимо пройти регистрацию, для этого необходимо нажать кнопку “Sign up” и далее следовать инструкциям на сайте для создания своего профиля. При наличии профиля в данной системе, необходимо нажать кнопку “Sign in” и пройти процесс авторизации вводя свои учётные данные. Процесс аутентификации представлен на рисунке 2.

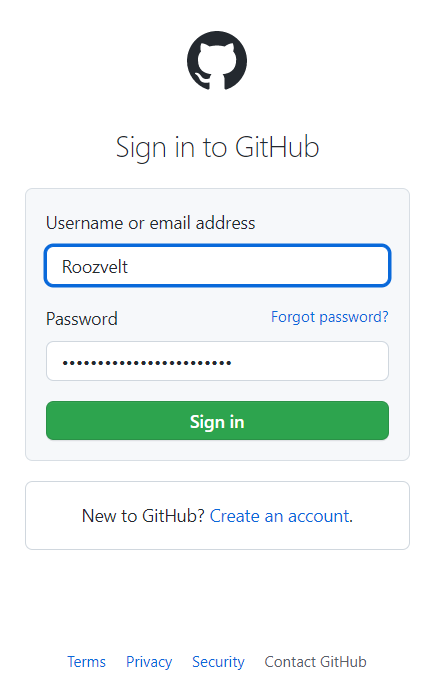


Рис. окно авторизации GitHub

После аутентификации в GitHub мы попадаем на главную страницу GitHub на которой мы можем использовать весь основной функционал данного сайта. Среди основных действий доступны:

* Создание своего репозитория
* Просмотр репозиториев других пользователей (если у них стоит флаг public)

Создать свой репозиторий в GitHub можно несколькими способами:

* Используя веб-интерфейс GitHub
* Используя приложения для ПК GitHub Desktop
* Используя любое другое приложение для работы с git с авторизацией в GitHub

Процесс создания репозитория через веб-интерфейс:

* для создания репозитория нужно нажать на + в правом верхнем углу сайта, выбрать пункт New Repository, заполнить название и описание, проставить нужные галочки и щёлкнуть на Create Repository;
  + Помимо интуитивно понятных названия и описания, необходимо указать тип доступа к репозиторию. Типа доступа существует два: Публичный и Приватный. Публичный репозиторий будет доступен всем в интернете. Другие пользователи смогут видеть все файлы, существующие в удалённом репозитории, и видеть процесс изменений данных файлов (историю версий). Приватный репозиторий будет виден только тем пользователям, которым будет предоставлен доступ до данного репозитория.
  + Указать файл README, это описание вашего проекта в нотации Markdown. Данное описание будет видно на главной страницы при открытии конкретного репозитория.
  + Добавить файл .gitignore в котором будет указаны те файлы, которые не должны попадать под изменений. Чаще всего сюда вносятся файлы создания локального окружения для работы с проектом, поскольку люди, работающие над проектом могут работать на различных машинах с использованием разных программ и соответственно создают для себя свои собственные окружения программной оболочки.
  + Выбрать лицензию распространения программного кода репозитория. Необходимо, если вы пишете программный код, который могут использовать пользователи в сети в своих целях, и вы хотите защитить его бесплатной лицензия по правилам который пользователи смогут использовать фрагменты программного кода или весь проект целиком.
* для загрузки файлов нужно зайти в нужный репозиторий, щёлкнуть на Add file и выбрать Upload files.

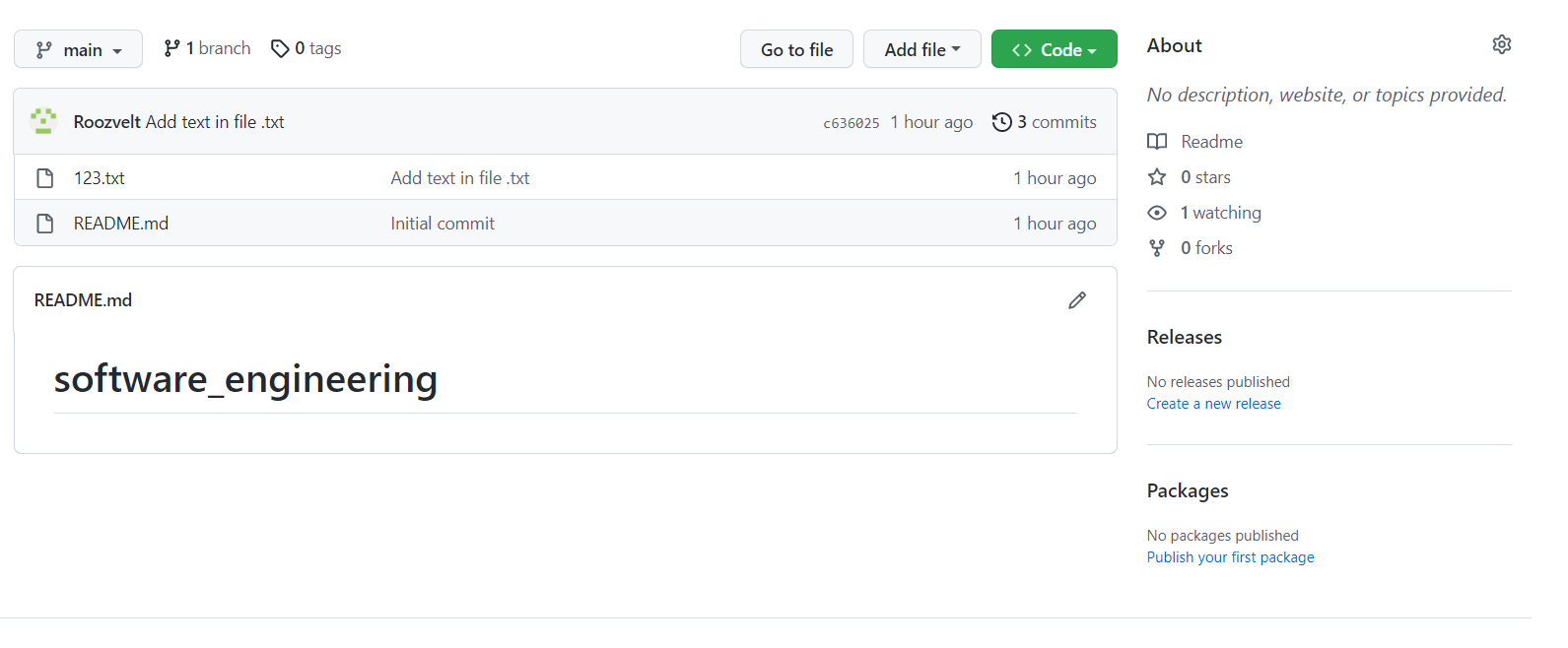


Рис. Главная страница репозитория

На главной странице репозитория будут отображены файлы, которые были загружены на удалённый репозиторий и доступные сообществу в интернете в зависимости от настроек вашего проекта. Под списком файлов в проекте будет выведена справочная информация, которая содержится в файле README.md. В правой части сайта указаны данные статистики по количеству просмотр, созданных форков и оценок вашего репозитория другими пользователями. Также будет информация о процентном соотношении используемых языков программирования в программного коде. Помимо этого, на главной странице репозитория можно посмотреть другие ветки разработки отличные от ветки master (main)

## 1.3 Основные команды GIT

Как упоминалось раннее для работы с GIT можно использовать различные инструменты: веб-инструменты, специализированные IDE с поддержкой git, или же использовать консольное приложение git for windows. Использование веб-инструмента или IDE с поддержкой удобно тем, что основные команды git там представлены использованием GUI интерфейса, что позволяет не прибегать к написанию команд git. Однако использование консольного приложения предоставляет наиболее полную возможность контроля работы с git. (Во всех известных IDE имеется возможность работы с git посредством консоли не используя интерфейс программы)

### 1.3.1 Создание репозитория

git init

Превращает обычный каталог в Git репозиторий чтобы начать версионировать файлы в нём.

### 1.3.2 Клонирование репозитория

git clone

git clone работает как обёртка над некоторыми другими командами. Она создаёт новый каталог, переходит внутрь и выполняет git init для создания пустого репозитория, затем она добавляет новый удалённый репозиторий (git remote add) для указанного URL (по умолчанию он получит имя origin), выполняет git fetch для этого репозитория и, наконец, извлекает последний коммит в ваш рабочий каталог, используя git checkout.

### 1.3.3 Добавления файлов для коммита

git add

Команда git add добавляет содержимое рабочего каталога в индекс (staging area) для последующего коммита. По умолчанию git commit использует лишь этот индекс, так что вы можете использовать git add для сборки слепка вашего следующего коммита.

### 1.3.4 Проверка состояния файлов в рабочем каталоге

git status

Команда git status показывает состояния файлов в рабочем каталоге и индексе: какие файлы изменены, но не добавлены в индекс; какие ожидают коммита в индексе. Вдобавок к этому выводятся подсказки о том, как изменить состояние файлов.

### 1.3.5 Просмотр изменений

git diff

Команда git diff используется для вычисления разницы между любыми двумя Git деревьями. Это может быть разница между вашей рабочей копией и индексом (собственно git diff), разница между индексом и последним коммитом (git diff --staged), или между любыми двумя коммитами (git diff master branchB).

### 1.3.6 Сохранение изменений

git commit

Команда git commit берёт все данные, добавленные в индекс с помощью git add, и сохраняет их слепок во внутренней базе данных, а затем сдвигает указатель текущей ветки на этот слепок.

### 1.3.7 Отмена изменений

git reset

Команда git reset, как можно догадаться из названия, используется в основном для отмены изменений. Она изменяет указатель HEAD и, опционально, состояние индекса. Также эта команда может изменить файлы в рабочем каталоге при использовании параметра --hard, что может привести к потере наработок при неправильном использовании, так что убедитесь в серьёзности своих намерений прежде чем использовать его.

### 1.3.8 Удаление файлов из индекса

git rm

Команда git rm используется в Git для удаления файлов из индекса и рабочей копии. Она похожа на git add с тем лишь исключением, что она удаляет, а не добавляет файлы для следующего коммита.

### 1.3.9 Перемещение файла

git mv

Команда git mv — это всего лишь удобный способ переместить файл, а затем выполнить git add для нового файла и git rm для старого.

### 1.3.10 Менеджер веток

git branch

Команда git branch — это своего рода "менеджер веток". Она умеет перечислять ваши ветки, создавать новые, удалять и переименовывать их.

### 1.3.11 Переключение веток

git checkout

Команда git checkout используется для переключения веток и выгрузки их содержимого в рабочий каталог.

### 1.3.12 Слияние веток

git merge

Команда git merge используется для слияния одной или нескольких веток в текущую. Затем она устанавливает указатель текущей ветки на результирующий коммит.

### 1.3.13 Просмотр истории коммитов

git log

Команда git log используется для просмотра истории коммитов, начиная с самого свежего и уходя к истокам проекта. По умолчанию, она показывает лишь историю текущей ветки, но может быть настроена на вывод истории других, даже нескольких сразу, веток. Также её можно использовать для просмотра различий между ветками на уровне коммитов.

### 1.3.14 Сбор изменений из удалённого репозитория

git fetch

Команда git fetch связывается с удалённым репозиторием и забирает из него все изменения, которых у вас пока нет и сохраняет их локально.

### 1.3.15 Слияния изменений из удалённого репозитория с локальным репозиторием

git pull

Команда git pull работает как комбинация команд git fetch и git merge, т. е. Git вначале забирает изменения из указанного удалённого репозитория, а затем пытается слить их с текущей веткой.

### 1.3.16 Отправление локальный изменений в удалённый репозиторий

git push

Команда git push используется для установления связи с удалённым репозиторием, вычисления локальных изменений отсутствующих в нём, и собственно их передачи в вышеупомянутый репозиторий. Этой команде нужно право на запись в репозиторий, поэтому она использует аутентификацию.