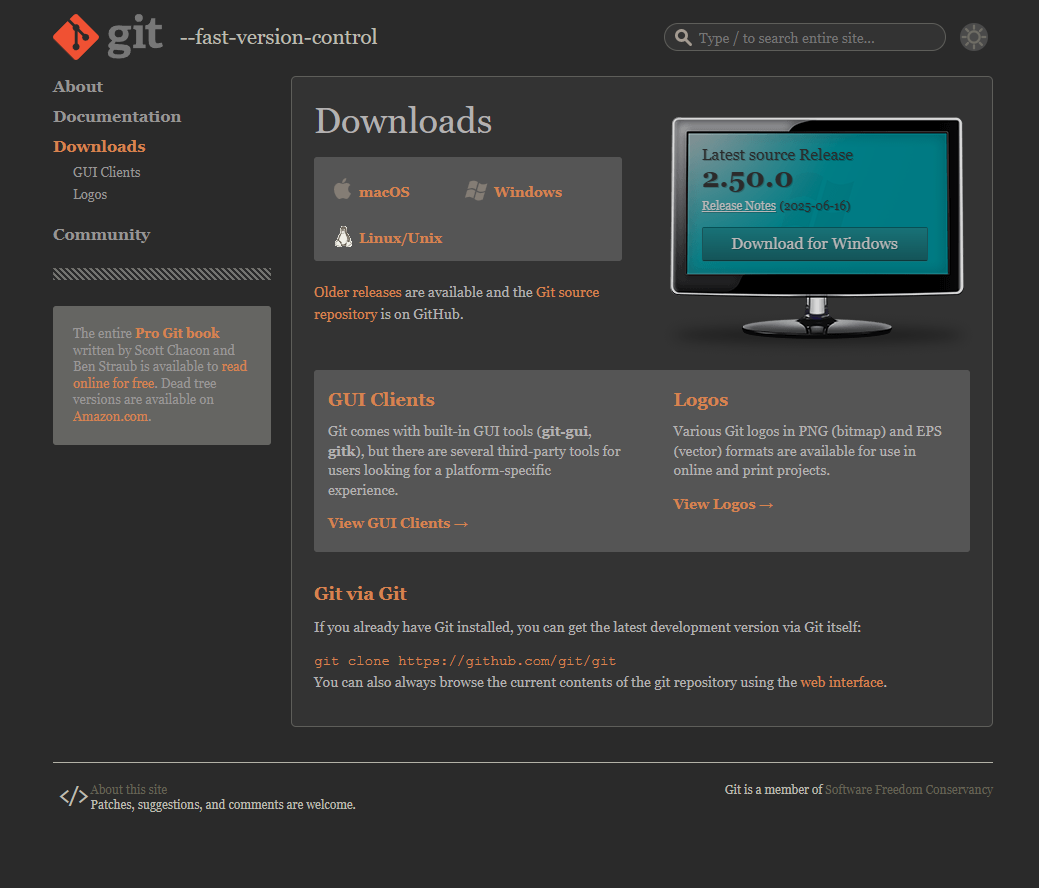
## Лабораторная работа по системе контроля версий git.

## 1. Переход на официальный сайт Git

Для начала я открыл браузер, установленный на моём компьютере (в моём случае это Google Chrome). В адресной строке браузера я ввёл официальный сайт Git — [**https://git-scm.com**](https://git-scm.com). После перехода на этот сайт открылся официальный ресурс разработчиков Git, где можно скачать последнюю актуальную версию программы.

Важно скачивать установочный файл именно с официального сайта, чтобы быть уверенным в безопасности и актуальности полученного программного обеспечения. На сайте представлена информация о доступных версиях Git для различных операционных систем: Windows, macOS и Linux. В моём случае система автоматически определила, что я работаю на Windows, и предложила скачать соответствующую версию.

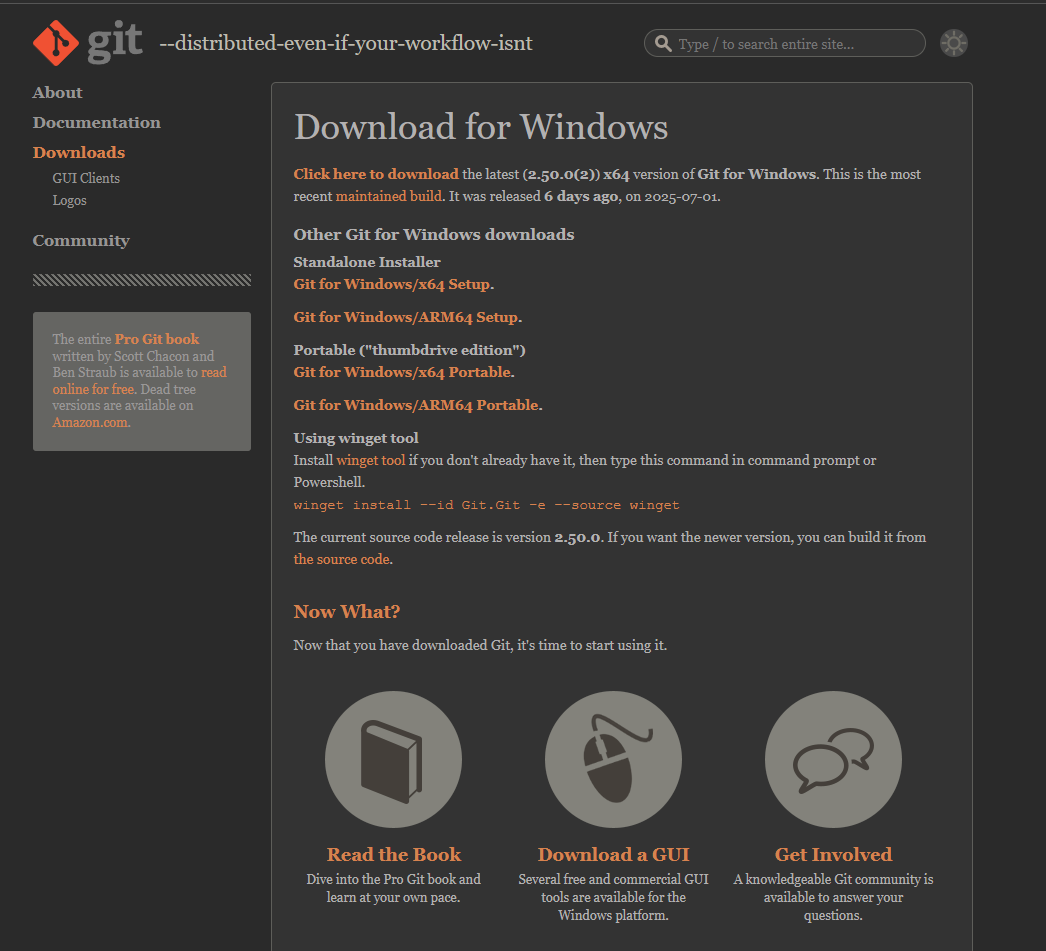


## 2. Скачивание установочного файла Git

После перехода на сайт в центре страницы я увидел большую кнопку **Download for Windows**, которая позволяет скачать установочный файл Git именно для моей операционной системы. Я нажал на эту кнопку левой кнопкой мыши, после чего началось скачивание установочного файла с расширением .exe.

В нижней части окна браузера появилось уведомление о загрузке. Я дождался, пока файл полностью скачался, прежде чем приступить к следующему шагу.

Скачивание установочного файла является важной частью процесса, так как именно этот файл содержит все необходимые компоненты для установки и дальнейшего использования Git.

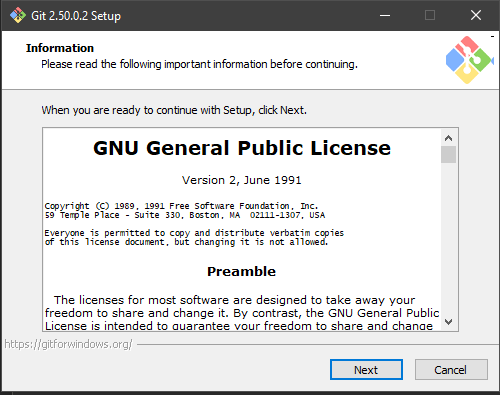


## 3. Запуск установочного файла

После завершения загрузки я открыл папку, в которую браузер сохранил установочный файл (по умолчанию это папка «Загрузки»). Дважды щёлкнул по установочному файлу **Git-setup.exe** для начала установки.

При запуске Windows может вывести предупреждающее окно контроля учётных записей, в котором нужно подтвердить запуск программы, нажав кнопку **Yes** (или «Да»). Это стандартная мера безопасности Windows при установке нового программного обеспечения.

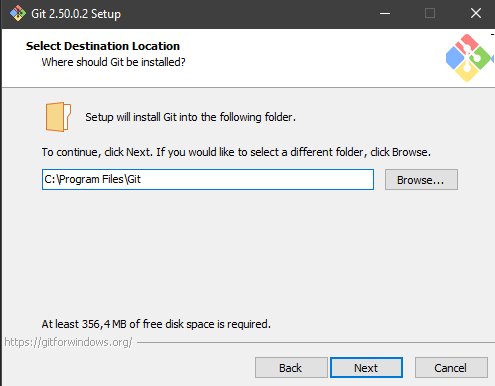
После запуска установщика открылось приветственное окно программы установки Git. В этом окне я нажал кнопку **Next**, чтобы перейти к следующему шагу. На данном этапе никаких изменений в настройках вносить не нужно — это просто начало процесса установки.

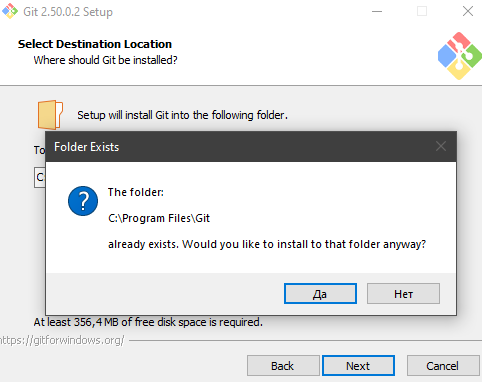


## 4.Выбор папки для установки

Далее установщик предложил выбрать папку, в которую будет установлена программа. По умолчанию Git устанавливается в папку C:\Program Files\Git, что является стандартной директорией для большинства программ на Windows.

Я оставил путь установки без изменений и нажал кнопку **Next** для продолжения. Если требуется, можно выбрать другую папку, но в большинстве случаев это не нужно.

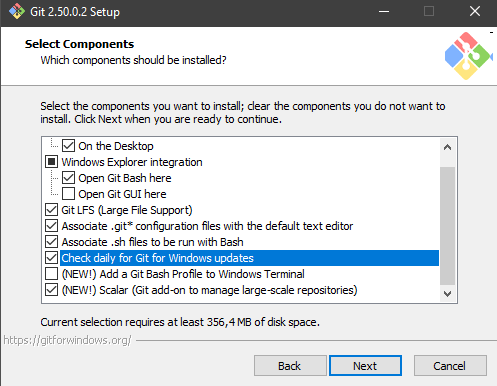


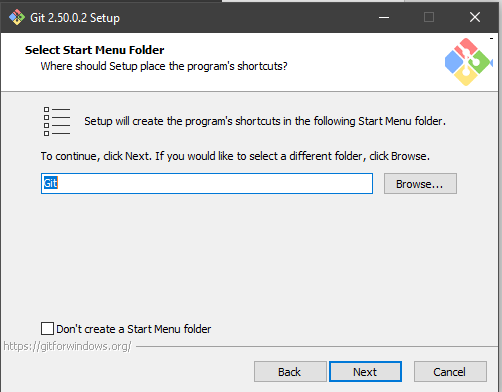


## 5.Выбор компонентов для установки

На следующем этапе установщик предложил выбрать компоненты, которые будут установлены. Здесь я ничего не менял и оставил все галочки по умолчанию, поскольку стандартные настройки подходят для нормальной работы Git.

Нажимаю кнопку **Next** для перехода дальше.

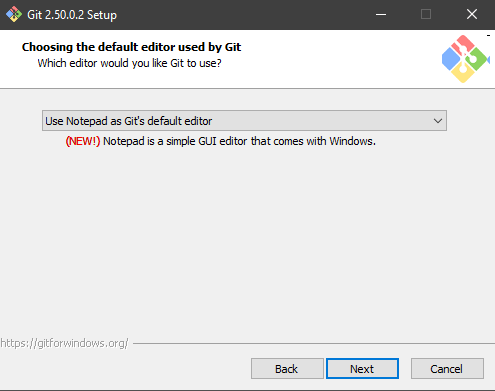




## 6.Выбор редактора по умолчанию

Далее установщик предложил выбрать текстовый редактор, который будет использоваться Git по умолчанию для написания сообщений коммитов. В списке был выбран стандартный редактор **Vim**, но так как он может быть неудобным для начинающих, я выбрал вариант с редактором **Visual Studio Code** (если он уже установлен).

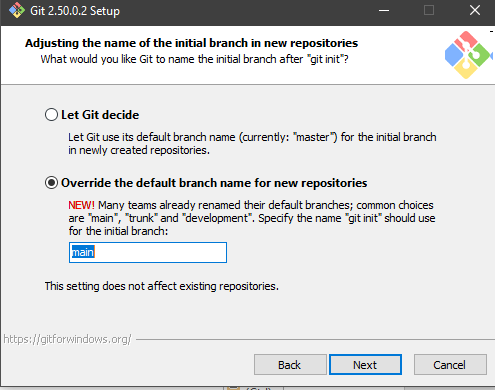
После выбора редактора нажимаю **Next**.



## 7.Настройка имени главной ветки (Initial Branch Name)

На этом этапе установки Git открылось окно, в котором нужно было выбрать имя главной ветки, которая будет автоматически создаваться при инициализации нового репозитория командой git init.

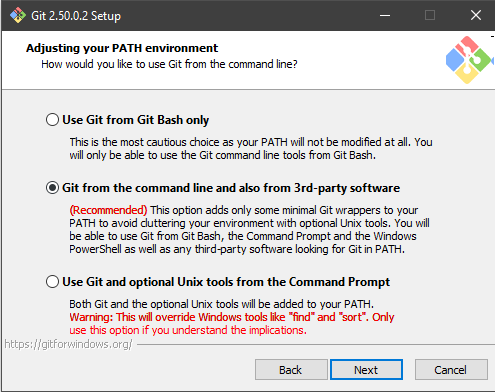
Я выбрал второй вариант — **Override the default branch name for new repositories** (Переопределить имя главной ветки для новых репозиториев). Это позволяет мне самому задать имя главной ветки вместо того, чтобы оставлять выбор за программой.



## 8.Настройка переменной PATH

Установщик предложил выбрать, каким образом команда git будет доступна в командной строке Windows. Я выбрал рекомендованный вариант **Git from the command line and also from 3rd-party software**, который позволяет использовать Git как в терминале Git Bash, так и в стандартной командной строке Windows.

Нажимаю **Next** для продолжения.



## 9.Выбор SSH-клиента для Git

На этом этапе установщик предлагает выбрать программу, которая будет использоваться для подключения к удалённым репозиториям через **SSH** (Secure Shell). SSH используется для безопасного взаимодействия с такими сервисами, как **GitHub**, **GitLab**, **Bitbucket** и другие.

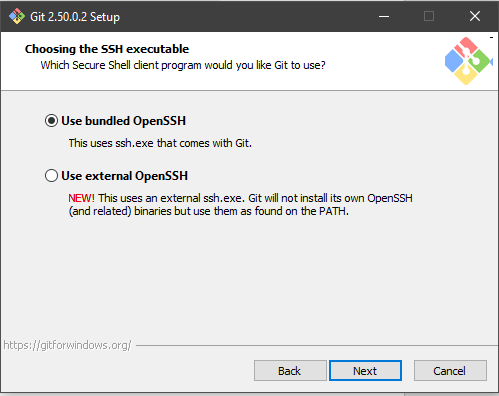
Установщик предлагает два варианта:

1. **Use bundled OpenSSH** — использовать встроенный SSH-клиент, который устанавливается вместе с Git.
2. **Use external OpenSSH** — использовать SSH-клиент, установленный в Windows отдельно.

Я оставил **первый вариант по умолчанию** — **Use bundled OpenSSH**.  
Это самый простой и надёжный способ, потому что встроенный OpenSSH сразу готов к работе, не требует дополнительных настроек и идеально подходит для большинства пользователей, особенно для начинающих.

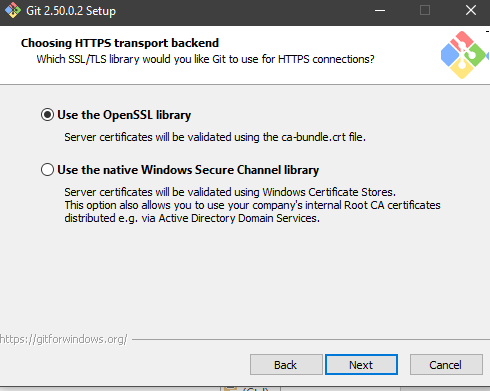
Это значит, что при работе с удалёнными репозиториями через SSH (например, при выполнении команд git clone, git push, git pull с SSH-ссылками) Git будет автоматически использовать этот внутренний клиент.

После выбора я нажал кнопку **Next**, чтобы продолжить установку



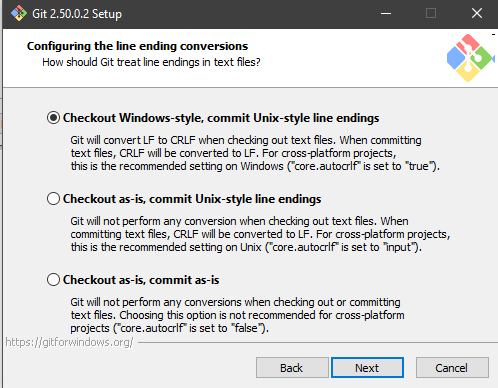
## 10.Настройка HTTPS-соединения

На следующем этапе установщик предложил выбрать способ проверки SSL-сертификатов при работе через HTTPS. Я оставил настройку по умолчанию и нажал **Next**.



## 11.Конфигурация окончания строк

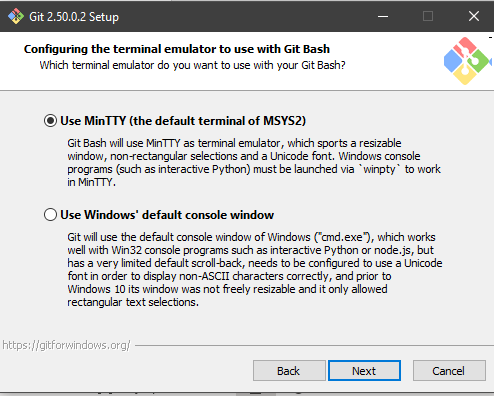
Затем установщик предложил выбрать способ обработки окончаний строк. Я оставил рекомендованный вариант **Checkout Windows-style, commit Unix-style line endings**. Это стандартная практика, которая позволяет работать с репозиториями без ошибок, связанных с разницей между системами.



## 12.Выбор терминала

Далее предлагалось выбрать терминал, который будет использоваться для запуска Git. Я оставил вариант **Use MinTTY (the default terminal of MSYS2)**. Это даёт возможность использовать более удобный терминал Git Bash.

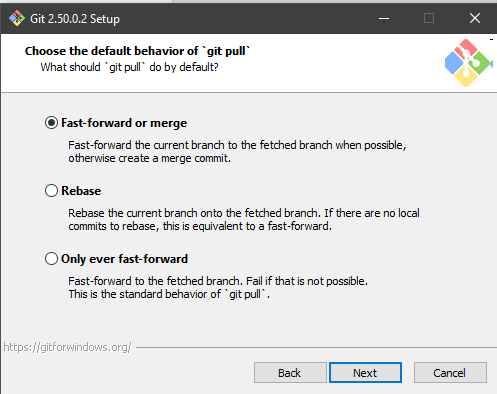
Нажимаю **Next**.

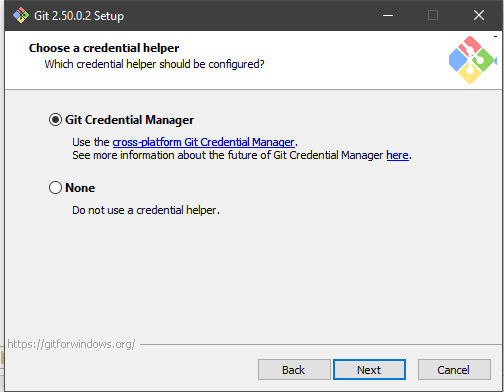


## 13.Дополнительные параметры

На следующем экране предлагались дополнительные параметры, касающиеся поведения Git. Я оставил все параметры по умолчанию, так как они подходят для большинства задач.

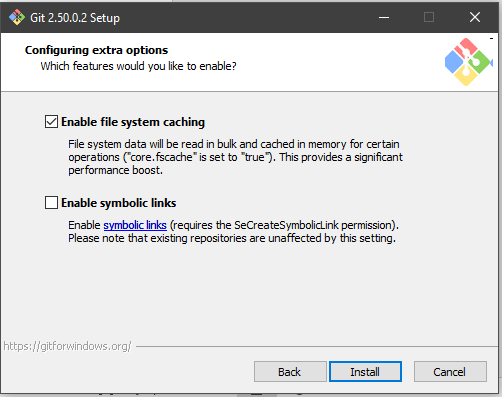
Нажимаю **Next**.





## 14.Запуск установки

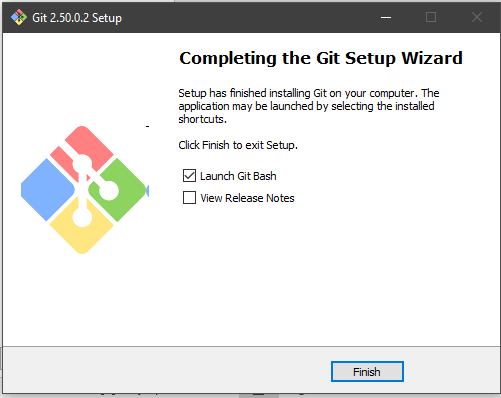
В следующем окне нажимаю кнопку **Install**, чтобы начать установку. Установщик начал копировать необходимые файлы на компьютер. Процесс установки занял несколько минут



## 15.Завершение установки

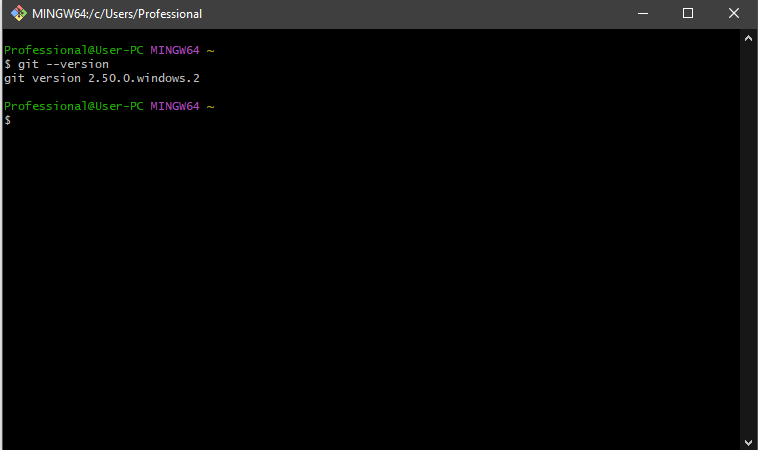
Когда установка завершилась, появилось последнее окно с кнопкой **Finish**. Я оставил галочку на пункте **Launch Git Bash** для того, чтобы сразу после установки открыть терминал.

Нажимаю **Finish**.



**16.Проверка успешной установки**

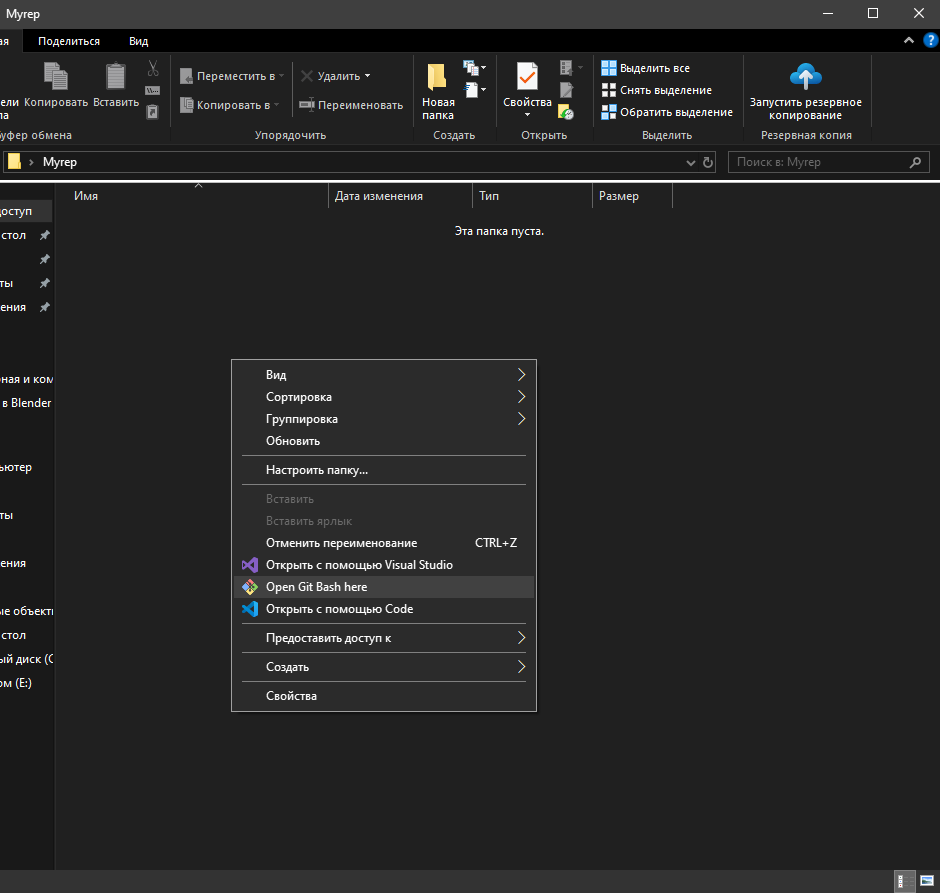
После завершения установки открылся терминал **Git Bash**. В этом терминале я ввёл команду: git –version.  
  
В ответ появилась версия установленного Git. Это подтверждает, что программа установлена правильно и готова к использованию.



### **17.Создание рабочей папки и запуск Git Bash**

Сначала на рабочем столе была создана папка с названием **Myrep**. Эта папка будет использоваться как рабочее пространство для проекта. Я зашёл внутрь этой папки и кликнул **правой кнопкой мыши** по пустому месту. В открывшемся меню выбрал пункт **Git Bash Here**.

Эта опция позволяет открыть командную строку **Git Bash** именно внутри выбранной папки, чтобы все команды выполнялись в нужной директории. Это важно, потому что Git отслеживает изменения именно в той папке, где он инициализирован.



**18.Инициализация локального репозитория**

Эта команда создаёт внутри папки скрытую папку **.git** и превращает обычную папку в локальный репозиторий Git. Теперь все изменения, происходящие внутри этой папки, могут отслеживаться и управляться с помощью Git.

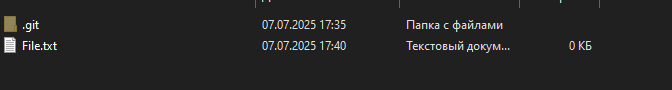
На экране появляется сообщение **Reinitialized existing Git repository in...**, что подтверждает успешную инициализацию. Этот шаг обязателен, чтобы начать работу с системой контроля версий.



### **19.Создание первого файла в репозитории**

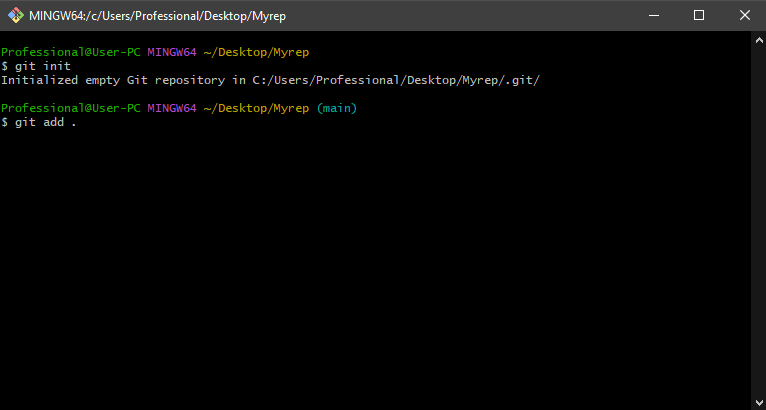
Для демонстрации дальнейшей работы я создал внутри папки **Myrep** простой текстовый файл с именем **File.txt**. Это можно сделать любым удобным способом: через проводник Windows или через редактор.

Этот файл нужен для того, чтобы показать, как Git отслеживает изменения. Пока файл создан, но Git его не видит и не отслеживает, об этом мы узнаем позже с помощью команды **git status**.



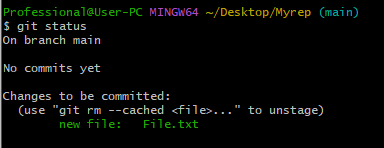
**20.Добавление всех файлов в индекс Git**

Точка означает «все файлы в текущей папке». Эта команда сообщает Git, что я хочу начать отслеживать все новые и изменённые файлы — они попадают в так называемую staging area (область подготовленных к коммиту файлов). Пока файлы не добавлены в индекс, Git не будет фиксировать их изменения.



**21.Проверка состояния репозитория**

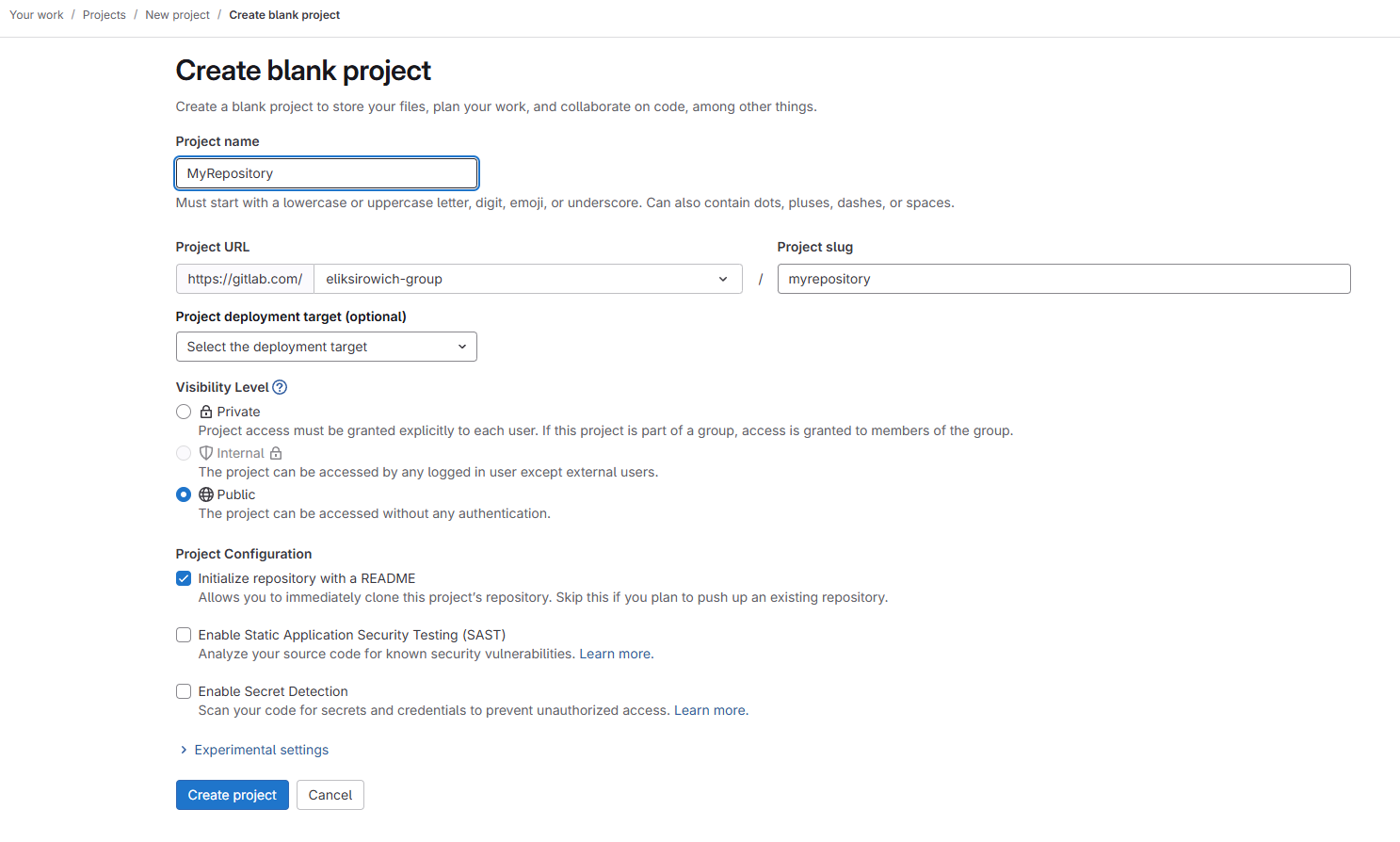
Git показал, какие файлы подготовлены к коммиту, какие ещё не отслеживаются, и общее состояние ветки. Это полезно, чтобы убедиться, что я не пропустил важные файлы перед фиксацией изменений.



### **22.Создание репозитория на GitLab**

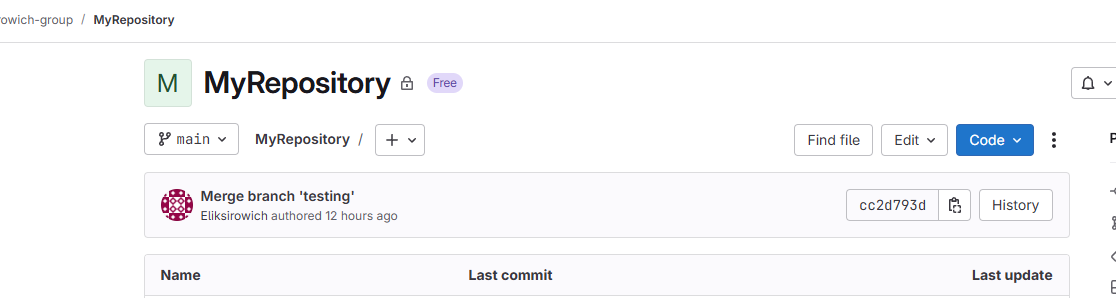
Затем я перешёл на сайт **github.com**, вошёл в свой аккаунт и нажал кнопку **New repository**. В форме создания указал название **MyRepository**, выбрал публичный доступ (Public), поставил галочку напротив **Add a README file** — чтобы автоматически создать файл README в удалённом репозитории. После этого нажал кнопку **Create repository**.

Созданный репозиторий — это пустое хранилище в интернете, куда можно загружать файлы с локального компьютера для хранения и совместной работы.



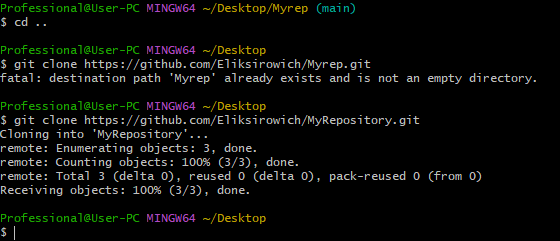
### **23.Получение ссылки на удалённый репозиторий**

После создания репозитория на GitLab я открыл страницу этого репозитория и нажал зелёную кнопку **Code**. В появившемся окне выбрал протокол HTTPS и скопировал адрес репозитория. Этот URL понадобится для клонирования репозитория на компьютер.



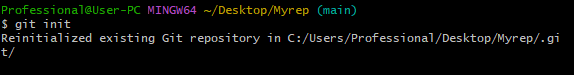
**24.Клонирование репозитория с GitLab**

Git скачал все файлы удалённого репозитория на мой компьютер, создав папку с именем репозитория. Это позволяет работать с актуальной версией проекта локально.



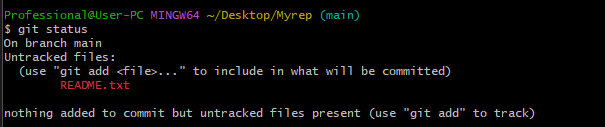
**25.Инициализация репозитория в клоне**

Git сообщил, что репозиторий уже существует, так как клонирование автоматически создаёт репозиторий с папкой **.git**. Этот шаг повторно подтверждает, что в текущей папке уже инициализирован Git.



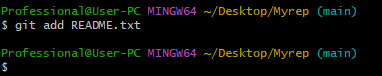
**26.Добавление нового файла README и проверка его статуса**

Я создал новый файл **README** (если его ещё не было) и выполнил команду. Git показал, что файл README не отслеживается, то есть он в статусе **Untracked files**. Это значит, что Git знает о файле, но ещё не отслеживает его изменения.



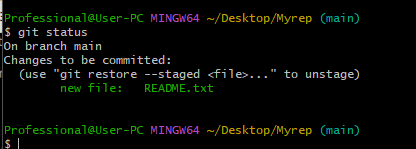
### **27.Добавление файла README в индекс**

Чтобы начать отслеживать изменения файла, ввёл команду. Теперь файл попал в staging area и готов к коммиту.



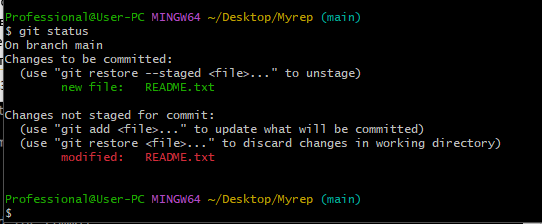
**28.Проверка статуса после добавления**

Git показал, что файл README теперь находится в состоянии **Changes to be committed**, то есть готов к фиксации.



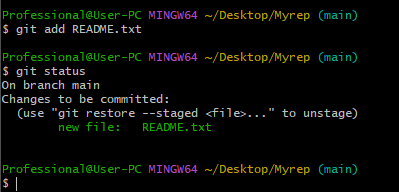
**29.Редактирование файла README и проверка изменений**

Я отредактировал файл README (внес изменения) и запустил. Git показал, что изменения не подготовлены к коммиту и находятся в секции **Changes not staged for commit**, а сам файл выделен красным. Это сигнал, что нужно добавить изменения в индекс.



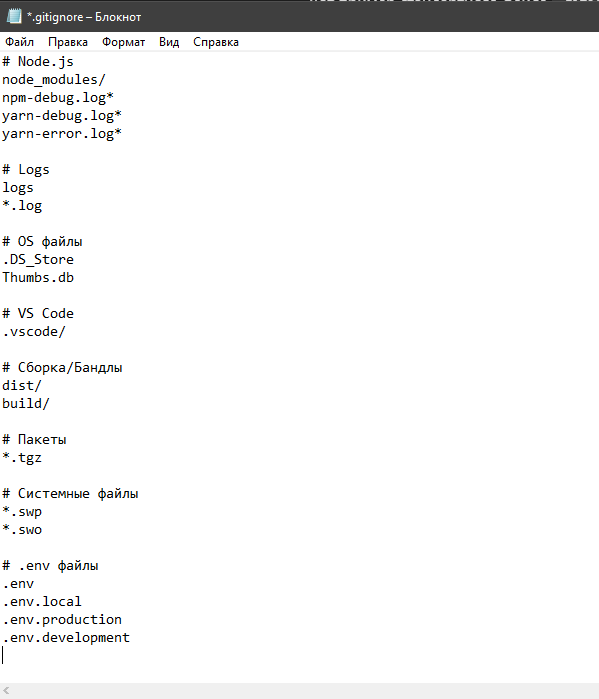
**30.Добавление изменений файла в индекс**

Теперь изменения подготовлены для коммита и будут сохранены при следующем коммите.



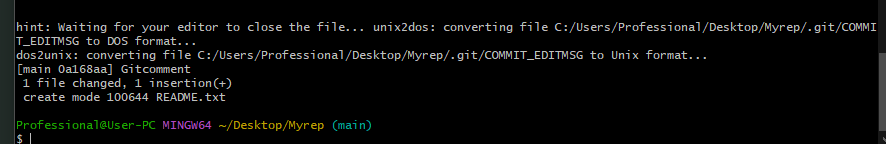
**31.Создание файла .gitignore и его содержание**

Создал в корне репозитория файл с именем **.gitignore**. Внутри указал шаблоны, которые Git должен игнорировать. Это нужно, чтобы не загружать временные или большие папки в репозиторий, облегчая работу с проектом.



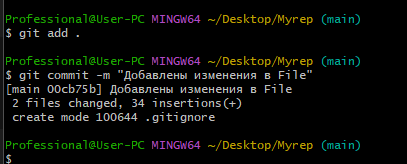
**32.Создание коммита с описанием**

Открылся редактор, в котором написал развернутое сообщение к коммиту, объясняющее суть изменений. Это позволяет сохранить текущую версию проекта с понятным описанием для истории.



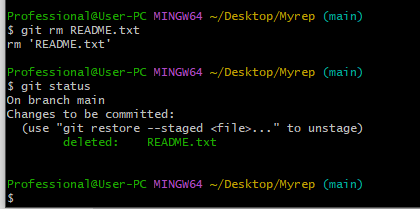
### **33.Быстрый коммит с сообщением**

Альтернативно, чтобы не открывать редактор, выполнил команду. Команда сразу создаёт коммит с указанным текстом, что экономит время при мелких изменениях.



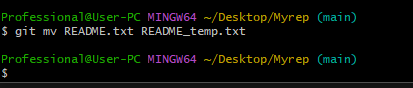
**34.Удаление файла README и проверка**

Удалил файл, затем снова проверил статус. Git показал, что файл удалён и ожидает фиксации.



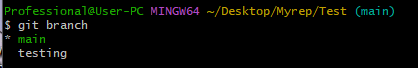
**35.Переименование файла**

Переименовал файл. Git зафиксировал это как удаление старого и добавление нового файла. Для сохранения изменений нужно выполнить коммит.



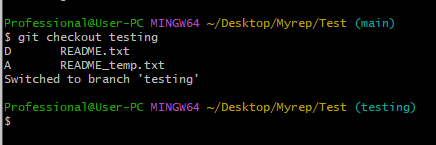
**36.Создание и просмотр веток**

Создал новую ветку с именем **testing**. Проверил все ветки. Отобразились ветки **main** и **testing**.



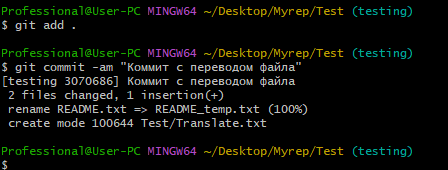
**37.Переключение на ветку testing**

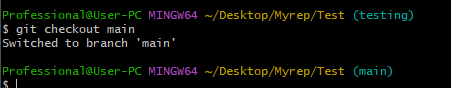
Git переключился на новую ветку, позволяя работать с отдельной историей изменений.



**38.Добавление и коммит изменений одной командой**

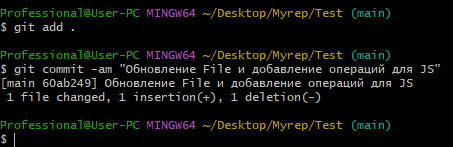
Так я добавил и закоммитил все изменения с одним сообщением.





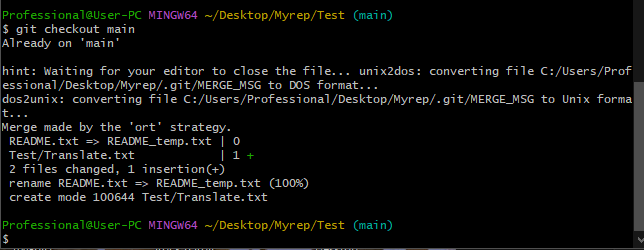
**39.Коммит с дополнительными изменениями.**

Сохранил ещё один набор изменений.

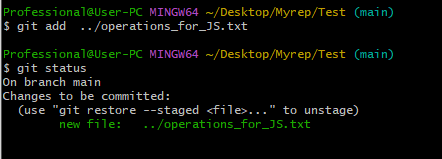


**40.Слияние веток**

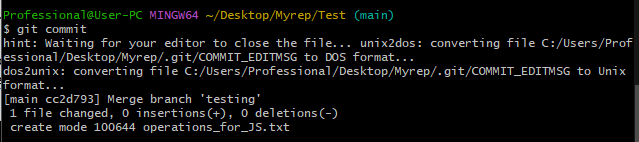
Созданную раннее ветку testing сливаем с веткой main



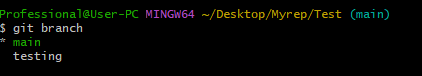
**41.Проверка**  
Проверяем статус ветки после слияния



**42.Коммит**  
Делаем новый коммит уже после слияния веток чтобы зафиксировать изменения.

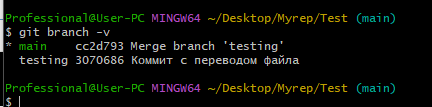


**43.Проверка**  
Проверяем рабочие ветки, если около ветки нет ‘\*’ значит можно стирать ветку так как она нам больше не нужна.



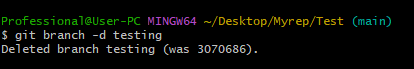
**44.Просмотр веток с последними коммитами**

Проверил последние коммиты в каждой ветке.



**45.Удаление ветки testing**

Удалил ветку. Git подтвердил удаление, освобождая пространство.



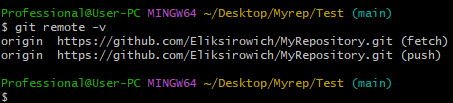
**46.Просмотр удалённых репозиториев**

Отобразился список подключённых удалённых репозиториев.



**47.Проверка URL удалённых репозиториев**

Команда показывает адреса для fetch и push.



**48.Получение изменений с удалённого репозитория.**

Обновил локальный список коммитов и веток без слияния.



**49.Отправка изменений на GitLab.**

Отправил локальные коммиты в удалённый репозиторий на ветку main.

