Лабораторная работа №1

Работа с git

Сахно Никита Вячеславович

1 Цель работы

Приобрести практические навыки работы с системой управления версиями Git.

2 Теоретическое введение

Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется. — В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять не полную версию изменённых файлов, а производить так называемую дельта-компрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. — Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить (слить) изменения, сделанные разными участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависимости от настроек блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, привилегированный доступ только одному пользователю, работающему с файлом. — Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например, они могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Кроме того, обычно доступна информация о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить.

Перечислим наиболее часто используемые команды git.

— Создание основного дерева репозитория:

git init

— Получение обновлений (изменений) текущего дерева из центрального репозитория:

git pull

— Отправка всех произведённых изменений локального дерева в центральный репозиторий:

git push

— Просмотр списка изменённых файлов в текущей директории:

git status

— Просмотр текущих изменений:

git diff

— Сохранение текущих изменений:

— Добавить все изменённые и/или созданные файлы и/или каталоги:

git add .

3 Выполнение лабораторной работы

3.1 Проверка наличия SSH и генерация ключевой пары

На начальном этапе была выполнена проверка работы утилиты SSH. При выполнении команды ssh был выведен список допустимых параметров, что подтверждает корректную установку SSH-клиента в системе (рис. 1).

```
nikit@nvsakhno MINGW64 ~
$ git config --global user.name "Сахно Никита"

nikit@nvsakhno MINGW64 ~
$ git config --global user.email "Nikita.Sakhno19@yandex.ru"

nikit@nvsakhno MINGW64 ~
$ git config --global core.quotepath false

nikit@nvsakhno MINGW64 ~
$ git config --global core.autocrlf true

nikit@nvsakhno MINGW64 ~
$ git config --global core.safecrlf warn

nikit@nvsakhno MINGW64 ~
$ git config --global init.defaultBranch master

nikit@nvsakhno MINGW64 ~
$ git config --global --list
user.name=Сахно Никита
user.email=Nikita.Sakhno19@yandex.ru
core.quotepath=false
core.autocrlf=true
core.safecrlf=warn
init.defaultbranch=master
```

Рисунок 1 — Проверка работы команды ssh.

Далее была выполнена генерация новой пары SSH-ключей с использованием криптографического алгоритма **ed25519** следующей командой:

ssh-keygen -t ed25519 -C "Nikita.Sakhno19@yandex.ru"

В результате были созданы закрытый и открытый ключи (id_ed25519 и id_ed25519.pub). Успешное завершение процесса генерации ключей отображено в терминале (рис. 2).

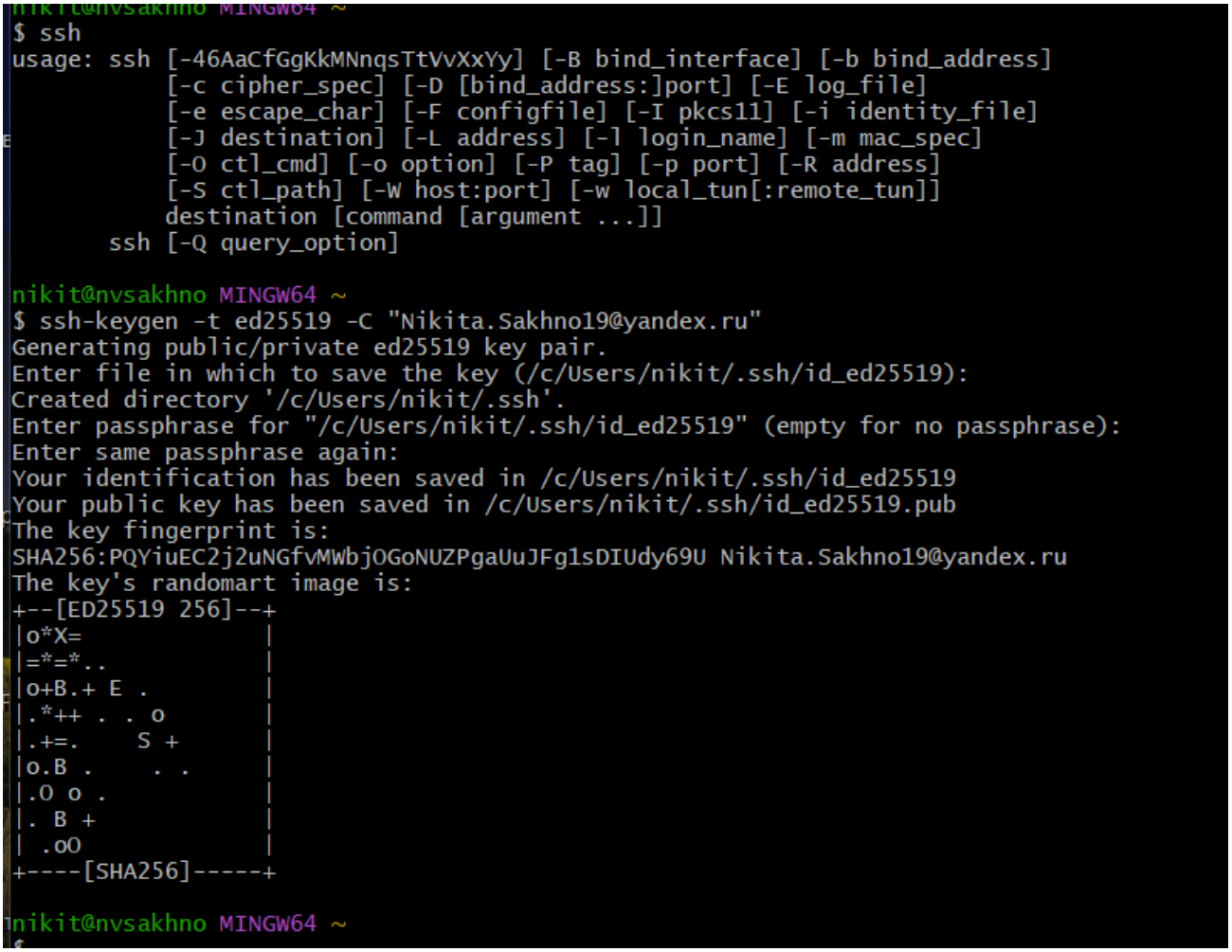


Рисунок 2 — Генерация SSH-ключей с использованием ssh-keygen.

3.2 Добавление SSH-ключа в GitHub и проверка аутентификации

Для добавления SSH-ключа в учётную запись GitHub было выведено содержимое публичного ключа с помощью команды:

cat ~/.ssh/id_ed25519.pub

После этого ключ был добавлен в настройках аккаунта GitHub в разделе **SSH keys**, что подтверждается его отображением в списке ключей (рис. 3).

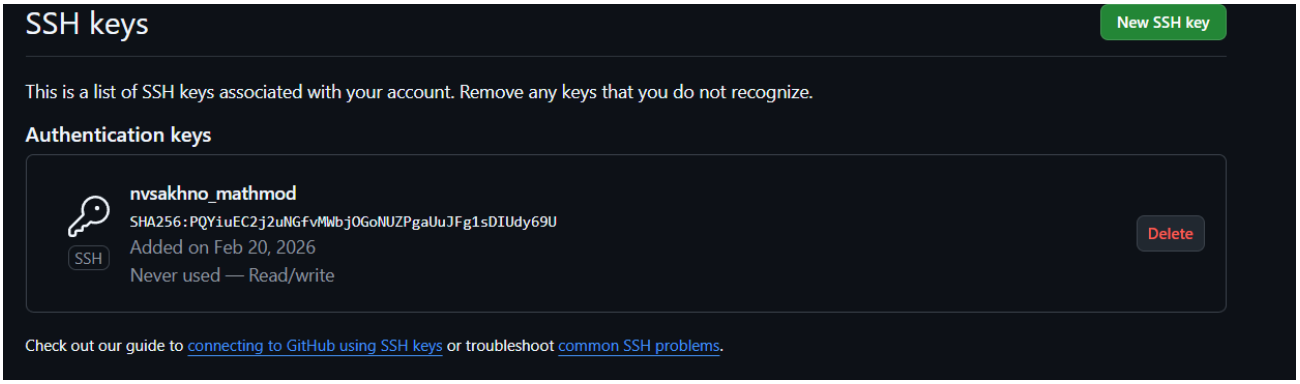


Рисунок 3 — Добавление SSH-ключа в настройках GitHub.

Для проверки корректности подключения к GitHub по SSH была выполнена команда:

ssh -T git@github.com

В ответ было получено сообщение об успешной аутентификации пользователя, что свидетельствует о корректной настройке SSH-доступа (рис. 4).

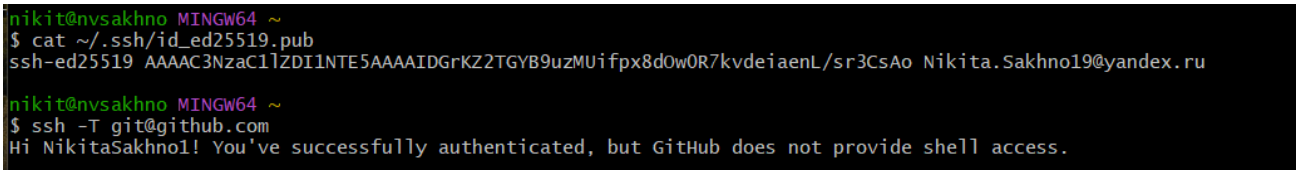


Рисунок 4 — Проверка SSH-подключения к GitHub.

3.3 Клонирование удалённого репозитория

После настройки SSH-доступа был выполнен процесс клонирования удалённого репозитория с использованием протокола SSH:

git clone git@github.com:yamadharma/course-directory-student-template.git

В результате репозиторий был успешно загружен на локальный компьютер, что подтверждается выводом сообщений о получении объектов (рис. 5).

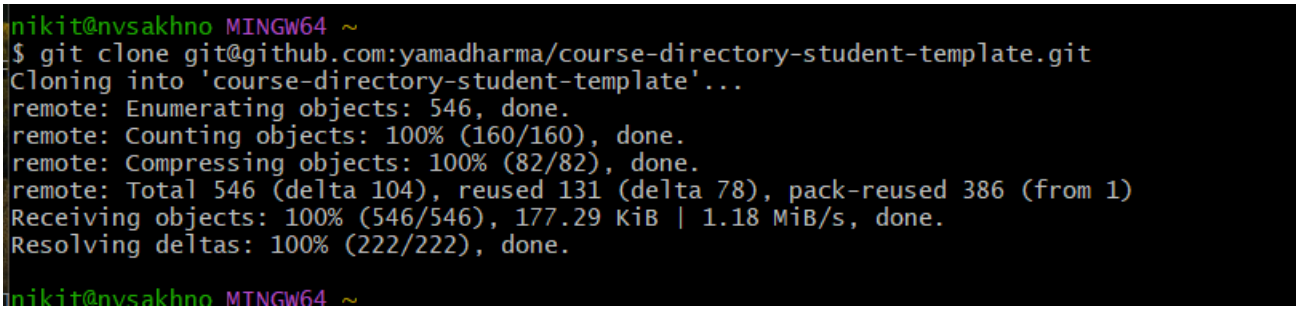


Рисунок 5 — Процесс клонирования удалённого репозитория.

Затем был выполнен переход в каталог проекта и просмотр его содержимого:

cd course-directory-student-template

ls

Команда ls отобразила структуру файлов репозитория (рис. 6).

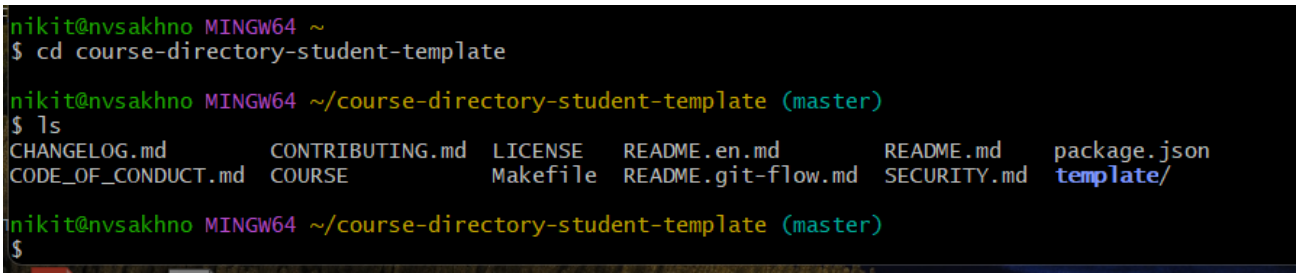


Рисунок 6 — Содержимое каталога репозитория после клонирования.

3.4 Создание файла COURSE

В соответствии с требованиями лабораторной работы в корне репозитория был создан файл COURSE, содержащий название курса. Файл был создан и заполнен командой:

echo simulation-modeling > COURSE

Корректность создания файла была проверена выводом его содержимого:

cat COURSE

В результате в файле содержится требуемое значение (рис. 7).

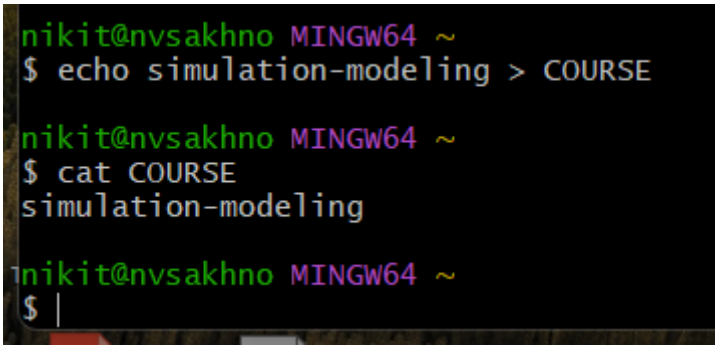


Рисунок 7 — Создание и проверка содержимого файла COURSE.

3.5 Настройка глобальной конфигурации Git

Для корректной идентификации пользователя и настройки параметров работы Git была выполнена глобальная конфигурация системы контроля версий. Были заданы имя пользователя, адрес электронной почты и параметры обработки файлов:

git config --global user.name "Сахно Никита"

git config --global user.email "Nikita.Sakhno19@yandex.ru"

git config --global core.quotepath false

git config --global core.autocrlf true

git config --global core.safecrlf warn

git config --global init.defaultBranch master

После этого выполнена проверка текущей конфигурации Git:

git config --global --list

Вывод команды подтверждает корректность установленных параметров (рис. 8).

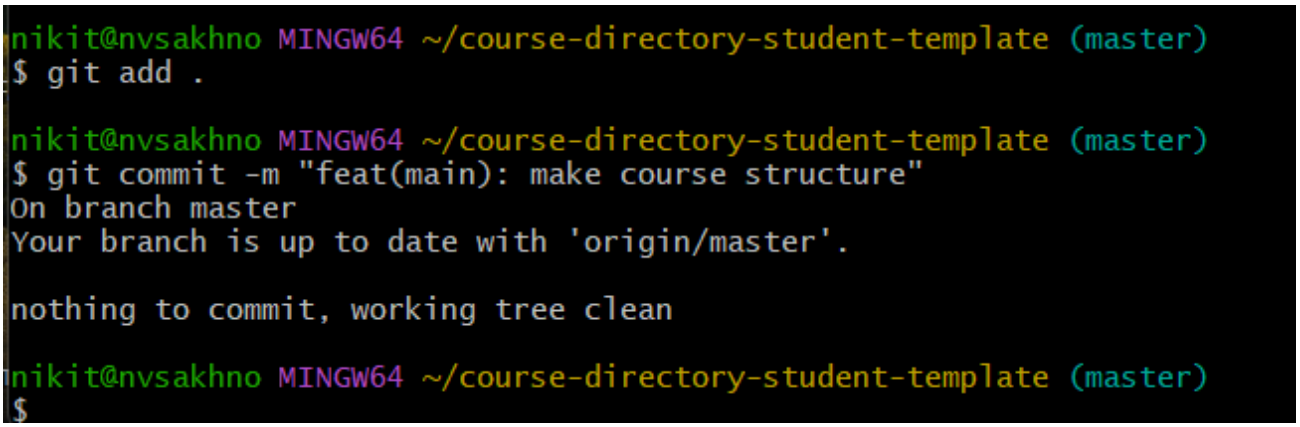


Рисунок 8 — Просмотр глобальной конфигурации Git.

3.6 Проверка версии git-flow

Для управления процессом ветвления был использован инструмент **git-flow (AVH Edition)**. Для проверки корректности установки была выполнена команда:

git flow version

В результате была отображена версия **1.12.3 (AVH Edition)**, что подтверждает готовность инструмента к работе (рис. 9).

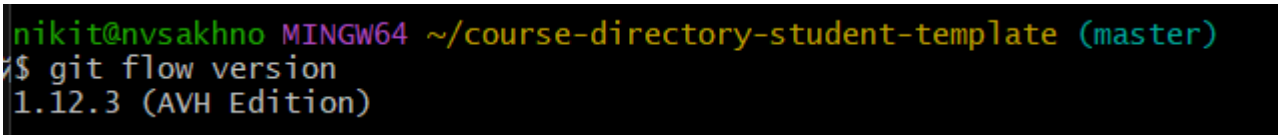


Рисунок 9 — Проверка версии git-flow.

3.7 Инициализация git-flow

Далее была выполнена инициализация git-flow в репозитории:

git flow init

В процессе инициализации были приняты значения по умолчанию, а в качестве префикса тега версии установлен символ v. По завершении инициализации была автоматически создана ветка develop (рис. 10).

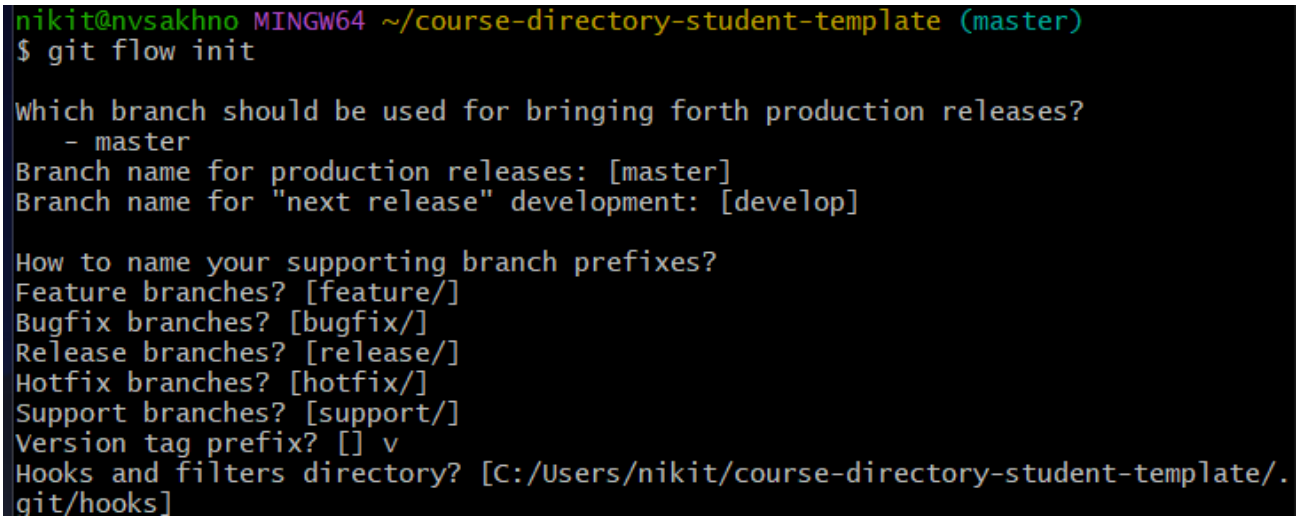


Рисунок 10 — Инициализация git-flow.

3.8 Проверка структуры ветвления

Для проверки структуры ветвления репозитория была выполнена команда:

git branch

В результате были отображены ветки master и develop, при этом активной является ветка develop (рис. 11).

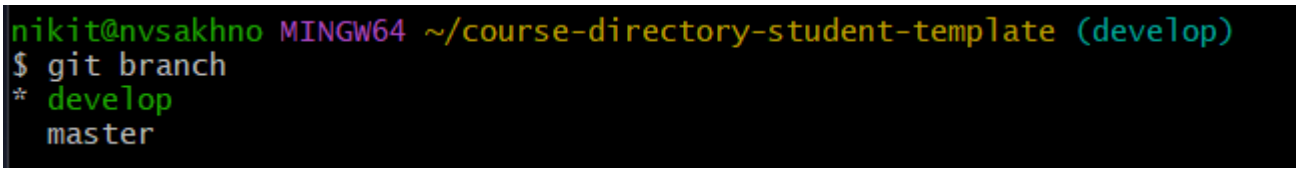


Рисунок 11 — Список веток репозитория после инициализации git-flow.

3.9 Создание релиза версии 1.0.0

Для подготовки стабильной версии проекта был начат процесс создания релиза:

git flow release start 1.0.0

В результате была создана ветка release/1.0.0, предназначенная для финальной подготовки релиза (рис. 12).

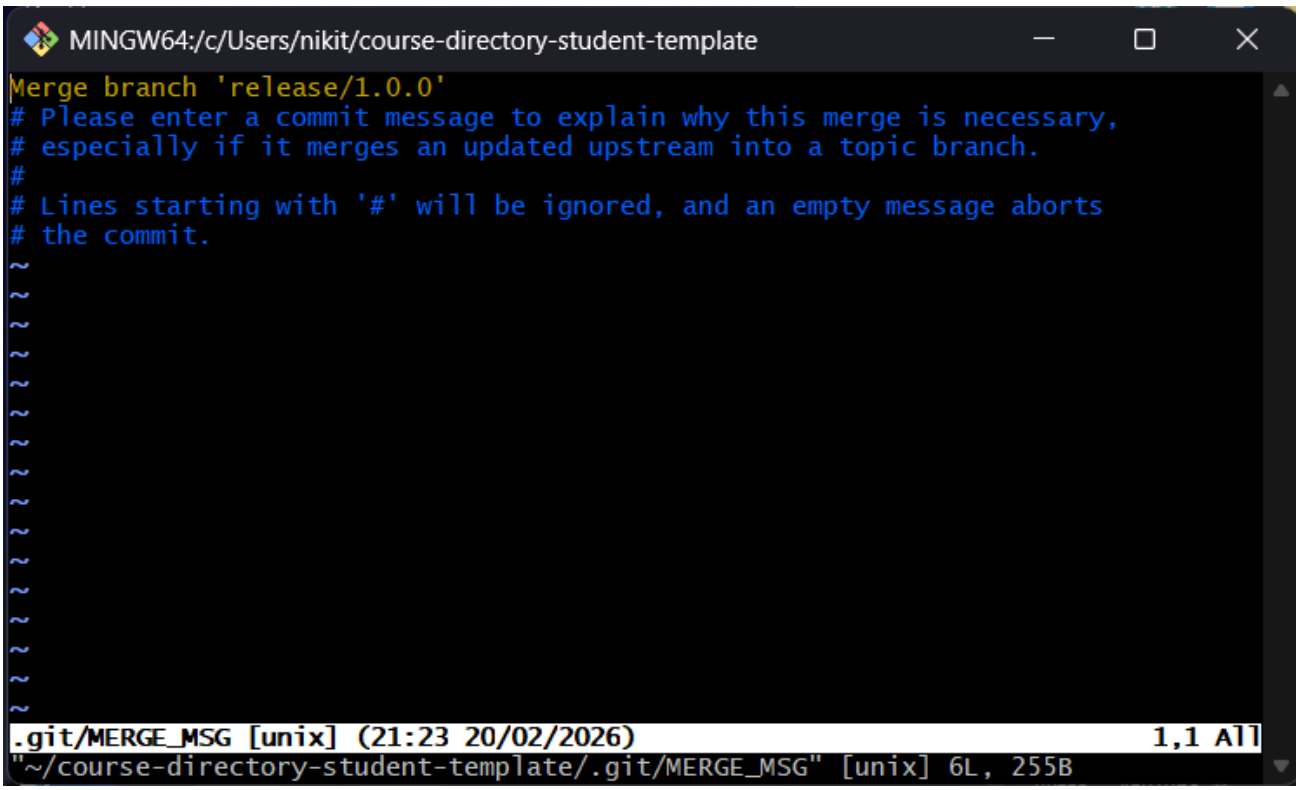


Рисунок 12 — Начало релиза версии 1.0.0.

3.10 Завершение релиза

После завершения подготовки релиза была выполнена команда:

git flow release finish 1.0.0

В ходе выполнения команды был автоматически открыт текстовый редактор **Vim** для подтверждения сообщения merge-коммита. После сохранения сообщения и выхода из редактора процесс завершения релиза был успешно завершён.

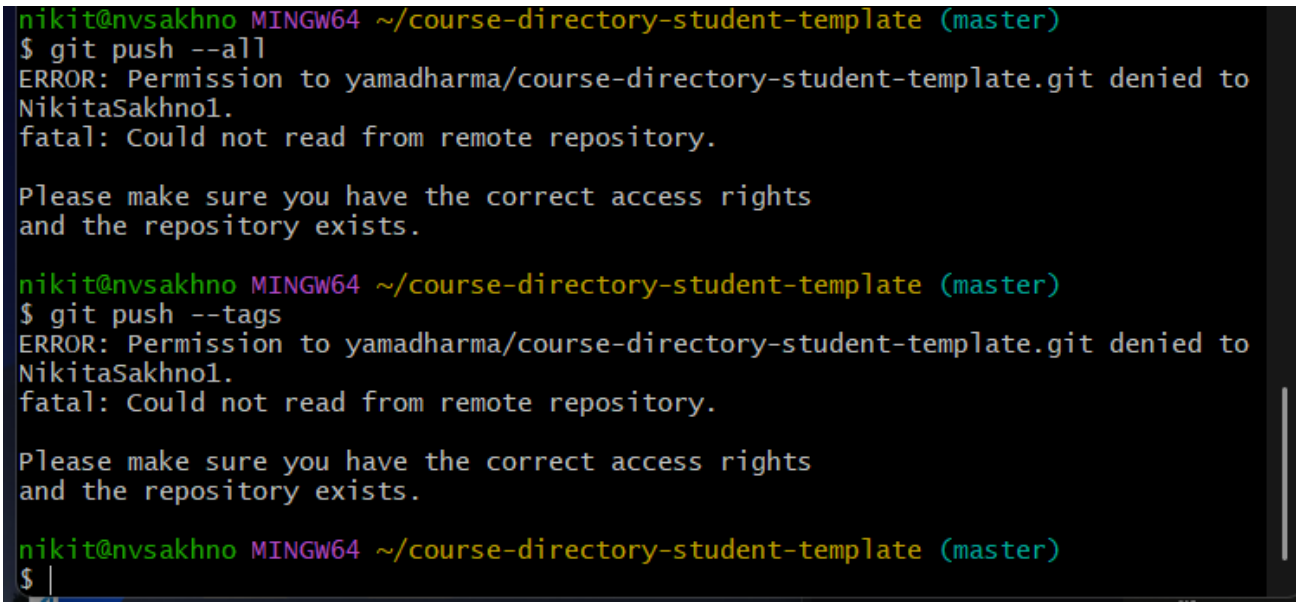
3.11 Отправка веток и тегов в удалённый репозиторий

Для синхронизации локального репозитория с удалённым были выполнены команды:

git push --all

git push --tags

Данные команды обеспечили отправку всех веток и тега версии v1.0.0 в удалённый репозиторий (рис. 13).



3.12 Проверка наличия тега версии

Для подтверждения успешного выпуска релиза была выполнена команда:

git tag

В списке тегов присутствует тег v1.0.0, что подтверждает корректное завершение процесса выпуска версии (рис. 14).

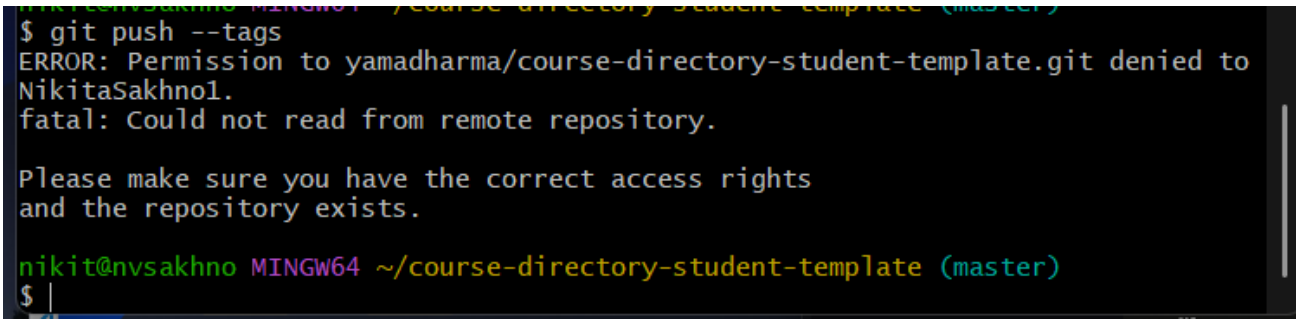


Рисунок 15 — Проверка наличия тега версии v1.0.0.

4 Выводы

В процессе выполнения данной лабораторной работы я приобрел практические навыки работы с Git.