МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

**Факультет информационных технологий и робототехники**

Кафедра программного обеспечения информационных систем и технологий

**Отчет**

**по лабораторной работе № 1**

**«Знакомство с системой контроля версий Git»**

**по дисциплине: «Конструирование программного обеспечения»**

Выполнили**:** ст.гр. 10701321

Мацкевич М.Ю.

Господарик П.В.

Принял: ст. преподаватель Станкевич С.Н.

Минск 2023

# Цель

Получить базовые знания и навыки при работе с системой контроля версий Git

# Введение

Git — абсолютный лидер по популярности среди современных систем управления версиями. Это развитый проект с активной поддержкой и открытым исходным кодом. Система Git была изначально разработана в 2005 году Линусом Торвальдсом — создателем ядра операционной системы Linux. Git применяется для управления версиями в рамках колоссального количества проектов по разработке ПО, как коммерческих, так и с открытым исходным кодом. Система используется множеством профессиональных разработчиков программного обеспечения. Она превосходно работает под управлением различных операционных систем и может применяться со множеством интегрированных сред разработки (IDE).

Git — система управления версиями с распределенной архитектурой. В отличие от некогда популярных систем вроде CVS и Subversion (SVN), где полная история версий проекта доступна лишь в одном месте, в Git каждая рабочая копия кода сама по себе является репозиторием. Это позволяет всем разработчикам хранить историю изменений в полном объеме.

Разработка в Git ориентирована на обеспечение высокой производительности, безопасности и гибкости распределенной системы.

# История создания git

Изначально Линус Торвальдс не использовал вообще никакой системы контроля версий. Люди, принимавшие участие в разработке ядра Linux, сначала должны были постить свои патчи в группе Usenet, а позже – отправлять по email. Все эти изменения Линус лично применял в своем дереве исходного кода. В итоге он выпускал новый релиз со всем этим деревом, без всякого разделения на какие-либо патчи. Единственным способом проследить историю этого процесса было изучение гигантского diff между двумя полными релизами.

Эта ситуация не была обусловлена отсутствием систем контроля версий в принципе. [CVS](https://ru.wikipedia.org/wiki/CVS) существовала еще с 1980-х и по-прежнему была самой популярной из такого рода систем. Она позволяла контрибуторам отправлять патчи в центральный репозиторий и изучать историю патчей в этом репозитории.

Но на CVS было много жалоб. Одной из проблем было то, что CVS отслеживала изменения «пофайлово» и не воспринимала более обширные патчи в качестве отдельной единой ревизии. Из-за этого разработчикам было тяжело отслеживать предыдущие изменения. Также в CVS были баги, которые было сложно исправить, например, [состояние гонки](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5_%D0%B3%D0%BE%D0%BD%D0%BA%D0%B8), когда одновременно отправлялись два конфликтующих патча.

Линусу не нравилась CVS. Частично по причинам, которые озвучивали другие люди, частично по собственным, которые стали понятны гораздо позже. Также ему не нравился [Subversion](https://ru.wikipedia.org/wiki/Subversion) – open-source проект, всплывший в 2000-х. Его целью было заменить CVS, исправив баги и недостатки последней.

Отсутствие подходящей системы контроля версий сильно огорчало многих разработчиков ядра Linux. Поэтому существенная часть сообщества давила на Линуса, побуждая его выбрать какой-то один из доступных вариантов. И вот, в 2002 году, выбор был сделан. К всеобщему ужасу и шоку, Линус выбрал [BitKeeper](https://www.bitkeeper.org/) – коммерческую систему с закрытым исходным кодом, разрабатываемую компанией BitMover под руководством Ларри МакВоя.

Ядро Linux было самым важным open-source проектом в истории. Линус Торвальдс был человеком, впервые открывшим подходы к open-source разработке, которые будут применяться в open-source проектах целыми десятилетиями, вплоть до сегодняшнего дня. О чем он только думал? Как он мог предать свое сообщество и мир открытого исходного кода? Именно эти мысли приходили в голову многим людям, когда Линус впервые начал использовать BitKeeper в разработке ядра.

Кроме того, BitMover в обмен на бесплатную лицензию установила существенные ограничения для Linux-сообщества. Во-первых, разработчикам не разрешалось использовать BitKeeper и при этом заниматься работой над системами контроля версий, которые могли бы стать конкурентами BitKeeper. А во-вторых, BitMover контролировала определенные метаданные, относящиеся к проекту ядра, чтобы отслеживать любые нарушения лицензии. Без доступа к этим метаданным разработчики не могли сравнивать прошлые версии ядер, а это было важным и уже стандартным функционалом других систем контроля версий.

Самым главным в BitKeeper была распределенная система, при помощи которой можно было с легкостью делать форки и мержить целые репозитории. Это было ключевое отличие системы. Теперь разработчики ядра могли сотрудничать внутри своей подгруппы, пользуясь всеми преимуществами контроля версий, а затем передавать свои уже готовые изменения Линусу. Таким образом, значительная часть работы, прежде лежавшая на плечах Линуса, распределялась между доверенными помощниками или даже членами любой группы. Архитектура, драйвера и подсистемы могли разрабатываться в некоторой степени независимо друг от друга, а затем одним большим «глотком» вливаться в главное дерево ядра.

Желание Линуса использовать несвободное ПО в самом сердце разработки ядра вдохновило многих людей на попытки создать альтернативу. CVS и Subversion не подходили на эту роль, поскольку отставали от BitKeeper, а кроме того имели фундаментальные ошибки в дизайне. То же касалось и других существовавших тогда проектов. Но поскольку все знали (или думал, что знали), чего хочет Линус, они могли подойти к делу со всей серьезностью. Результатом стала целая череда систем контроля версий, предлагавших распределенную разработку.

Спустя три года напряженных усилий ни одна open-source альтернатива не приблизилась к удовлетворению нужд Линуса ближе, чем CVS или Subversion. Такое положение вещей могло бы сохраняться еще долго, если бы не Эндрю Тридгелл, создатель Samba, участник создания rsync и вообще прекрасный парень.

В 2005 году Эндрю попытался воспроизвести сетевые протоколы BitKeeper для создания свободной альтернативы этого ПО. Причем даже если бы он этого не делал, это сделал бы кто-нибудь другой, это было просто делом времени.

Ларри МакВой предупреждал разработчиков Linux, что он им перекроет кислород, если кто-нибудь попытается сделать нечто подобное, и так он и поступил. Внезапно оказалось, что BitKeeper уже не может использоваться при разработке ядра. Вся цепочка инструментов разработки вертелась вокруг распределенной системы контроля версий, а теперь все это повисло в неопределенности.

Что это означало? Вернется ли Торвальдс к старой схеме разработки, лично принимая все патчи? Если нет, то будет ли выбран один из альтернативных вариантов с открытым кодом? И если да, то какой из них?

И тут случилось нечто выдающееся. Впервые с 1991 года Линус занялся чем-то помимо ядра Linux. Поскольку ни один из существовавших инструментов его не удовлетворял, он решил написать собственный.

Одной из приоритетных для Линуса вещей была скорость. Он никогда не говорил об этом, по крайней мере, не так внятно, чтобы кто-нибудь из разработчиков СКВ сделал на этом упор.

Поскольку разработчики ядра по всему миру полным ходом присылали свои патчи, Линусу нужно было что-то с невообразимой до тех пор скоростью. Он не мог позволить ожидание в несколько секунд даже для окончания самых больших и сложных операций. А ни arch, ни darcs, ни monotone, ни еще какой-нибудь проект даже и близко не удовлетворяли этому требованию.

Линус недолго пописал код в уединении, а затем представил миру свою новую концепцию. За несколько дней от начала проекта в июне 2005 года система контроля версий git стала полностью самостоятельной. Через несколько недель она уже была готова к размещению разработки ядра Linux. Через пару месяцев система стала полностью функциональной. На этом этапе Линус передал поддержку проекта его главному контрибутору-энтузиасту Джунио Хамано, а сам вновь вернулся к разработке Linux.

Потрясенное сообщество разработчиков свободного ПО изо всех сил пыталось разобраться в этом странном творении. Оно не походило ни на какие другие системы контроля версий. Фактически, оно скорее напоминало низкоуровневую файловую систему, чем СКВ.

Вместо того чтобы хранить патчи, как другие системы, [Git](https://techrocks.ru/2019/02/14/git-cheatsheet-for-beginners/) сохранял полные версии каждого измененного файла. И что в этом хорошего? А с другой стороны, он мог молниеносно обрабатывать форки и слияния, а также быстро генерировать патчи при необходимости.

Постепенно Джунио собрал набор команд более высокого уровня, которые больше походили на команды таких инструментов как CVS и Subversion. И если первоначальный набор команд git был скорее «канализацией», то новый можно было считать «фарфором». Так их и назвали. (*Канализация – plumbing, фарфор – porcelain. Под «фарфором» понимаются фарфоровые унитазы и раковины, т. е., то, с чем больше соприкасается человек, вещи более высокого уровня, чем канализационные трубы.* – прим. перев.).

Новая система контроля версий вызвала энтузиазм, соразмерный спорам и негодованию по поводу BitKeeper. Повсюду появлялись новые порты, расширения и сайты. Через несколько лет git пользовались уже практически все. Как и Linux, он захватил весь мир.

# Основные команды

## Как задать имя пользователя и адрес электронной почты

Имя пользователя нужно, чтобы привязывать коммиты к вашему имени. Это не то же самое, что имя пользователя учётной записи GitHub, с помощью которого выполняется вход в профиль на GitHub. Задать или изменить имя пользователя можно с помощью команды git config. Новое имя будет автоматически отображаться в последующих коммитах, отправленных на GitHub через командную строку. Если хотите скрыть своё реальное имя, можно использовать в качестве имени пользователя Git произвольный набор символов.

git config --global user.name "Tara Routray"

Кроме того, командой git config можно изменять адрес электронной почты, привязанный к вашим коммитам Git. Новый адрес электронной почты будет автоматически отображаться во всех дальнейших коммитах, поданных на GitHub через командную строку.

git config --global user.email "[dev@tararoutray.com](mailto:dev@tararoutray.com)"

## Инициализация репозитория

Создать пустой репозиторий Git или вновь инициализировать существующий можно параметром init. При инициализации он создаст скрытую папку. В ней содержатся все объекты и ссылки, которые Git использует и создаёт в истории работы над проектом.

git init

## Добавление отдельных файлов или всех файлов в область подготовленных файлов

Добавить отдельный файл в область подготовленных файлов можно параметром add с указанием имени файла. Просто замените somefile.js на актуальное имя.

git add somefile.js

Кроме того, можно добавить все файлы и папки в эту область, предоставив wildcard . вместо имени файла:

git add .

## Проверка статуса репозитория

Просмотреть статус нужного репозитория можно по ключевому слову status: его действие распространяется на подготовленные, неподготовленные и неотслеживаемые файлы.

git status

## Внесение изменений однострочным сообщением или через редактор

При создании коммита в репозитории можно добавить однострочное сообщение с помощью параметра commit с флагом -m. Само сообщение вводится непосредственно после флага, в кавычках.x

git commit -m "Your short summary about the commit"

Также можно открыть текстовый редактор в терминале для написания полного сообщения коммита. Оно может состоять из нескольких строк текста, в котором подробно характеризуются изменения, внесённые в репозиторий.

git commit

## Просмотр истории коммитов с изменениями

Просматривать изменения, внесённые в репозиторий, можно с помощью параметра log. Он отображает список последних коммитов в порядке выполнения. Кроме того, добавив флаг -p, вы можете подробно изучить изменения, внесённые в каждый файл.

git log -p

## Просмотр заданного коммита

Просмотреть полный список изменений, внесённых конкретным коммитом, можно с помощью параметра show, указав идентификатор или хеш коммита. Значение хеша уникально для каждого коммита, созданного в вашем репозитории.

git show 1af17e73721dbe0c40011b82ed4bb1a7dbe3ce29

Также можно использовать сокращённый хеш.

git show 1af17e

## Просмотр изменений до коммита

Можно просматривать список изменений, внесённых в репозиторий, используя параметр diff. По умолчанию отображаются только изменения, не подготовленные для фиксации.

git diff

Для просмотра подготовленных изменений необходимо добавить флаг --staged.

git diff --staged

Также можно указать имя файла как параметр и просмотреть изменения, внесённые только в этот файл.

git diff somefile.js

## Изменение последнего коммита

Внести изменения в последний коммит можно параметром commit с флагом --amend. Например, вы записали изменения, внесённые в ряд файлов, и поняли, что допустили ошибку в сообщении коммита. В этом случае можете воспользоваться указанной командой, чтобы отредактировать сообщение предыдущего коммита, не изменяя его снимок.

git commit --amend -m "Updated message for the previous commit"

Также можно вносить изменения в файлы, отправленные ранее. Например, вы изменили несколько файлов в ряде папок и хотите их записать как единый снимок, но забыли добавить в коммит одну из папок. Чтобы исправить такую ошибку, достаточно подготовить для фиксации остальные файлы и папки и создать коммит с флагами --amend и --no-edit.

git add dir1

git commit

# Here you forgot to add dir2 to commit, you can execute the

following command to amend the other files and folders.

git add dir2

git commit --amend --no-edit

Флаг --no-edit позволит внести в коммит поправку без изменения сообщения коммита. В этом случае итоговый коммит заменит неполный, а выглядеть это будет так, как будто мы отправили изменения ко всем файлам в нужных папках как единый снимок.

Внимание! Не изменяйте публичные коммиты.

С помощью amend прекрасно исправляются локальные коммиты, а исправления можно передать в общий репозиторий. Однако изменять коммиты, уже доступные другим пользователям, не следует. Помните, что изменённые коммиты являются совершенно новыми, а предыдущий коммит уже не будет доступен в текущей ветке. Последствия будут такими же, как при отмене изменений публичного снимка.

## Откат последнего коммита

Откатить последний коммит можно с помощью параметра revert. Создастся новый коммит, содержащий обратные преобразования относительно предыдущего, и добавится к истории текущей ветки.

git revert HEAD

### ▍ Разница между revert и reset

Команда git revert отменяет изменения, записанные только одним коммитом. Она не откатывает проект к более раннему состоянию, удаляя все последующие коммиты, как это делает команда git reset.

У команды revert есть два крупных преимущества по сравнению с reset. Во-первых, она не меняет историю проекта и производит операцию, безопасную для коммитов. Во-вторых, её объектом выступает конкретный коммит, созданный в любой момент истории, а git reset всегда берёт за точку отсчёта текущий коммит. К примеру, если нужно отменить старый коммит с помощью git reset, придётся удалить все коммиты, поданные после целевого, а затем выполнить их повторно. Следовательно, команда git revert — гораздо более удобный и безопасный способ отмены изменений.

## Откат заданного коммита

Откатить проект до заданного коммита можно с помощью параметра revert и идентификатора коммита. Создастся новый коммит — копия коммита с предоставленным идентификатором — и добавится к истории текущей ветки.

git revert 1af17e

## Создание новой ветки и переход в неё

Создать новую ветку можно с помощью параметра branch, указав имя ветки.

git branch new\_branch\_name

Но Git не переключится на неё автоматически. Для автоматического перехода нужно добавить флаг -b и параметр checkout.

git checkout -b new\_branch\_name

## Просмотр списка веток

Можно просматривать полный список веток, используя параметр branch. Команда отобразит все ветки, отметит текущую звёздочкой (\*) и выделит её цветом.

git branch

Также можно вывести список удалённых веток с помощью флага -a.

git branch -a

## Удаление ветки

Удалить ветку можно параметром branch с добавлением флага -d и указанием имени ветки. Если вы завершили работу над веткой и объединили её с основной, можно её удалить без потери истории. Однако, если выполнить команду удаления до слияния — в результате появится сообщение об ошибке. Этот защитный механизм предотвращает потерю доступа к файлам.

git branch -d existing\_branch\_name

Для принудительного удаления ветки используется флаг -D с заглавной буквой. В этом случае ветка будет удалена независимо от текущего статуса, без предупреждений.

git branch -D existing\_branch\_name

Вышеуказанные команды удаляют только локальную копию ветки. В удалённом репозитории она может сохраниться. Если хотите стереть удалённую ветку, выполните следующую команду:

git push origin --delete existing\_branch\_name

## Слияние двух веток

Объединить две ветки можно параметром merge с указанием имени ветки. Команда объединит указанную ветку с основной.

git merge existing\_branch\_name

Если надо выполнить коммит слияния, выполните команду git merge с флагом --no-ff.

git merge --no-ff existing\_branch\_name

Указанная команда объединит заданную ветку с основной и произведёт коммит слияния. Это необходимо для фиксации всех слияний в вашем репозитории.

## Прекращение слияния при конфликте

Прервать слияние в случае конфликта можно параметром merge с флагом --abort. Он позволяет остановить процесс слияния и вернуть состояние, с которого этот процесс был начат.

git merge --abort

Также при конфликте слияния можно использовать параметр reset, чтобы восстановить конфликтующие файлы до стабильного состояния.

git reset

## Отправка изменений в удалённый репозиторий

Отправлять изменения в удалённый репозиторий можно параметром push с указанием имени репозитория и ветки.

git push origin main

Эта команда передаёт локальные изменения в центральный репозиторий, где с ними могут ознакомиться другие участники проекта.

## Получение изменений из удалённого репозитория

Для загрузки изменений из удалённого репозитория используется параметр pull. Он скачивает копию текущей ветки с указанного удалённого репозитория и объединяет её с локальной копией.

git pull

Также можно просмотреть подробные сведения о загруженных файлах с помощью флага --verbose.

git pull --verbose

## Слияние удалённого репозитория с локальным

Слияние удалённого репозитория с локальным выполняется параметром merge с указанием имени удалённого репозитория.

git merge origin

## Отправка новой ветки в удалённый репозиторий

Передать новую ветку в удалённый репозиторий можно параметром push с флагом -u, указав имя репозитория и имя ветки.

git push -u origin new\_branch

## Удаление удалённой ветки

Чтобы избавиться от удалённой ветки, используйте параметр push с флагом --delete, указав имя удалённого репозитория и имя ветки.

git push --delete origin existing\_branch

## Использование перебазирования

Для доступа к этой функции используйте параметр rebase с указанием имени ветки. Перебазирование — это процесс объединения или перемещения последовательности коммитов на новый родительский снимок.

git rebase branch\_name

Эта команда изменит основу ветки с одного коммита на другой, как если бы вы начали ветку с другого коммита. В Git это достигается за счёт создания новых коммитов и применения их к указанному базовому коммиту. Необходимо понимать, что, хотя ветка и выглядит такой же, она состоит из совершенно новых коммитов.

Выполнение задания

Для выполнения задания мы выбрали платформу GitHub.

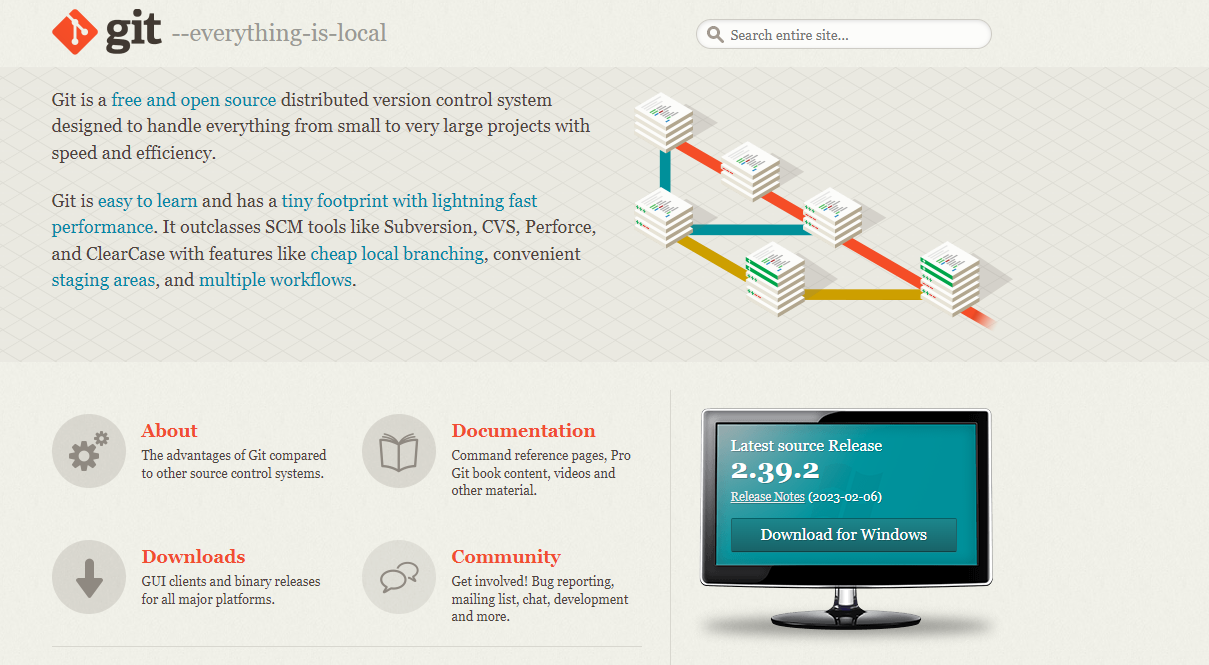


Рисунок 1 - сайт системы контроля версий Git

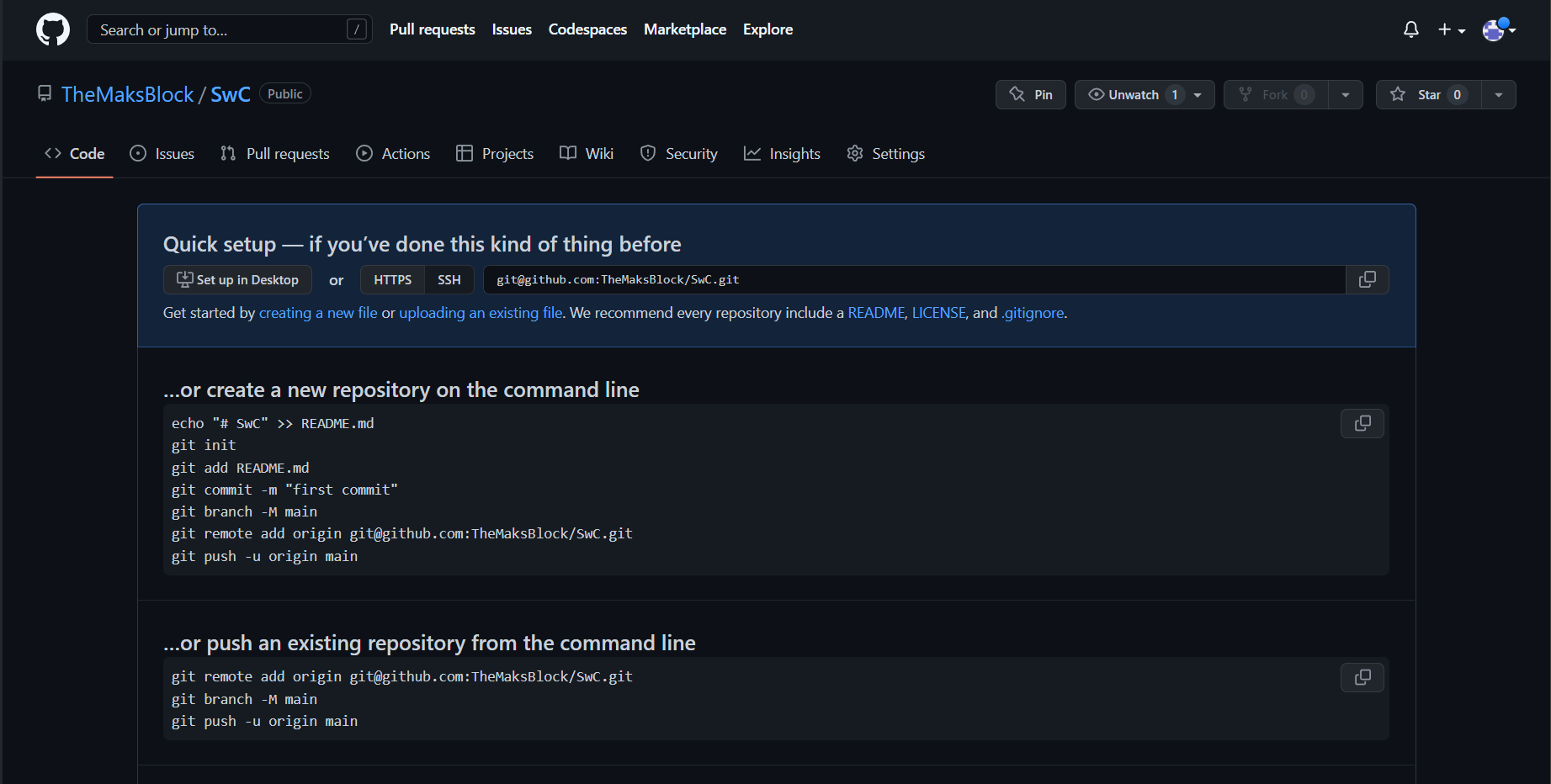


Рисунок 2 - окно созданного репозитория

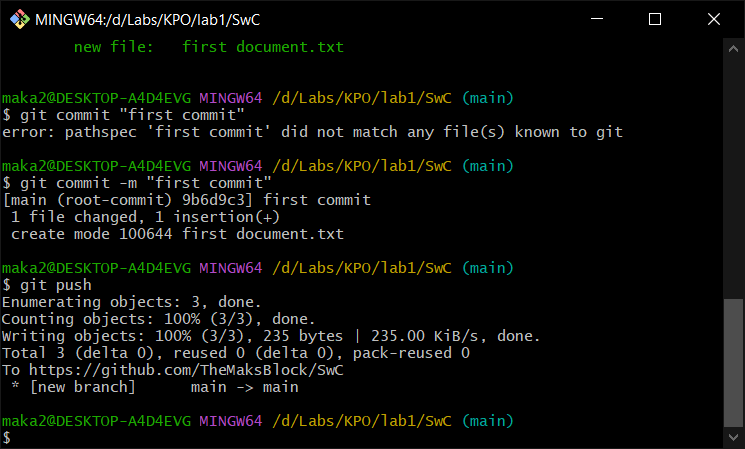


Рисунок 3 - скриншот Git Bash

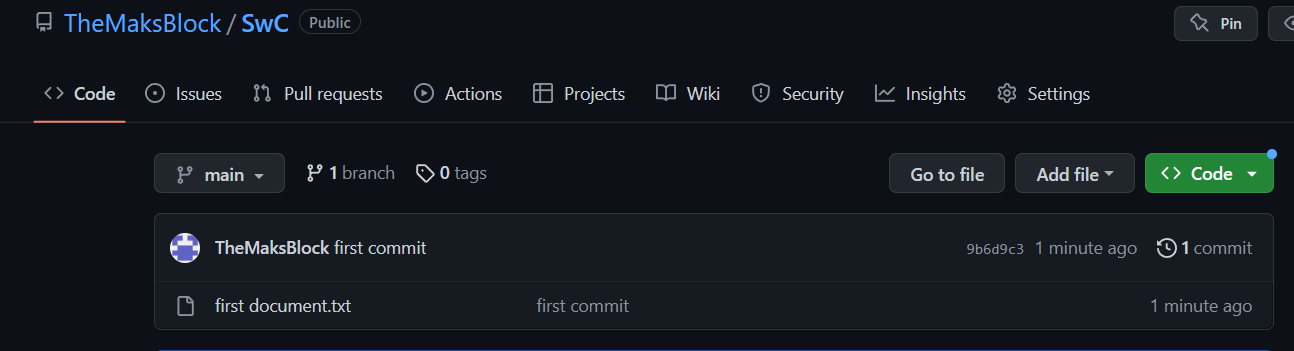


Рисунок 4 - результат выполнения лабораторной работы

# Вывод

В данной лабораторной работе мы получили знания и научились пользоваться системой контроля версий Git

# Список использованных источников

1.[Git - О системе контроля версий (git-scm.com)](https://git-scm.com/book/ru/v2/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B5-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8F-%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9)

2. <https://github.com>

3. [Git — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/Git)