Отчёт по лабораторной работе

Лабораторная №2

Полина Витальевна Барабаш

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Целью данной работы является изучить идеологию и применение средств контроля версий и освоить умения по работе с git.

2 Задание

- Создать базовую конфигурацию для работы с git.
- Создать ключ SSH.
- Создать ключ PGP.
- Настроить подписи git.
- Зарегистрироваться на Github.
- Создать локальный каталог для выполнения заданий по предмету.

3 Выполнение лабораторной работы

Задание 1. Установить git.

Я вошла в режим суперпользователя и установила git с помощью команды dnf install git (рис. [3.1]).

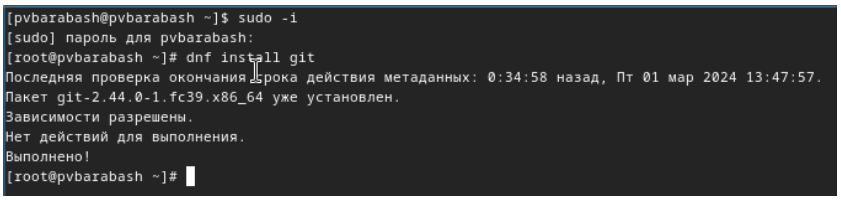

Рис. 3.1: Установка git

Задание 2. Установить gh.

Я установила gn с помощью команды dnf install gh (рис. [3.2]).

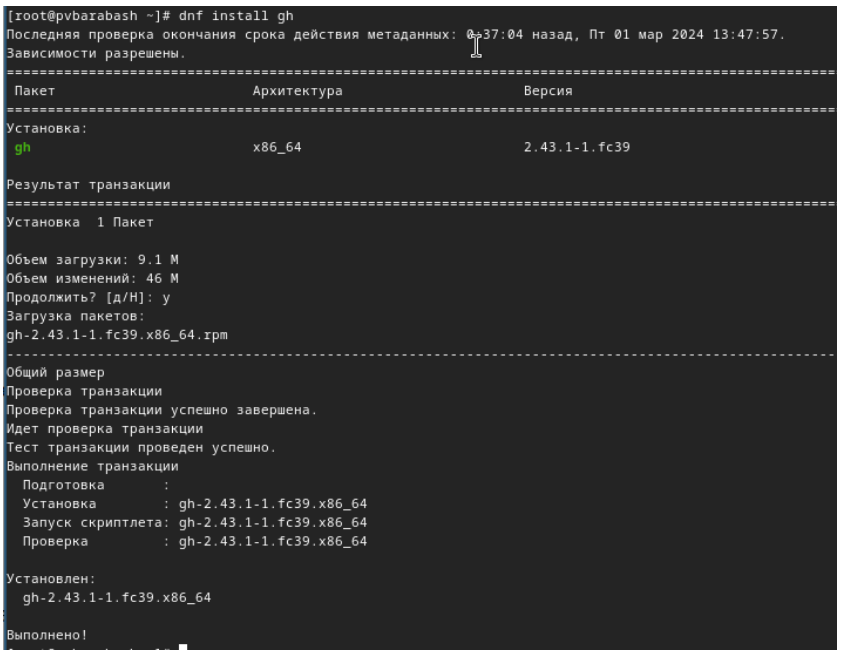

Рис. 3.2: Установка gh

Задание 3. Произвести базовую настройку git. Задать имя и email владельца репозитория. Настроить utf-8 в выводе сообщений git. Задать имя начальной ветки, параметр autocrlf, параметр safecrlf.

Я задала имя и email своего репозитория с помощью двух команд:

git config –global user.name “Name Surname”

git config –global user.email “work@mail”

Вместо Name Surname я ввела имя своего репозитория и вместо work@mail почту, на которую зарегистрирована учетная запись (рис. [3.3]).

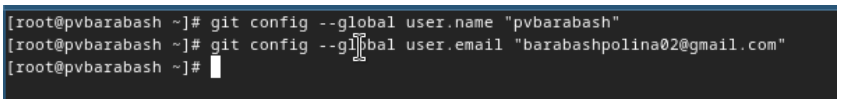

Рис. 3.3: Задание имени и emal репозитория

Я настроила utf-8 в выводе сообщений git с помощью команды git config –global core.quotepath false, задала имя начальной ветки master с помощью команды git config –global init.defaultBranch master, параметр autocrlf с помощью

git config –global core.autocrlf input и параметр safecrlf с помощью git config –global core.safecrlf warn (рис. [3.4]).

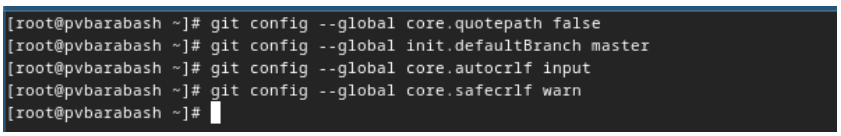

Рис. 3.4: Настройка utf-8, задание имени начальной ветки и параметров

Задание 4. Создать ключи ssh по двум алгоритмам.

Я создала ключ ssh по алгоритму rsa с ключём размером 4096 бит с помощью команды ssh-keygen -t rsa -b 4096 (рис. [3.5]).

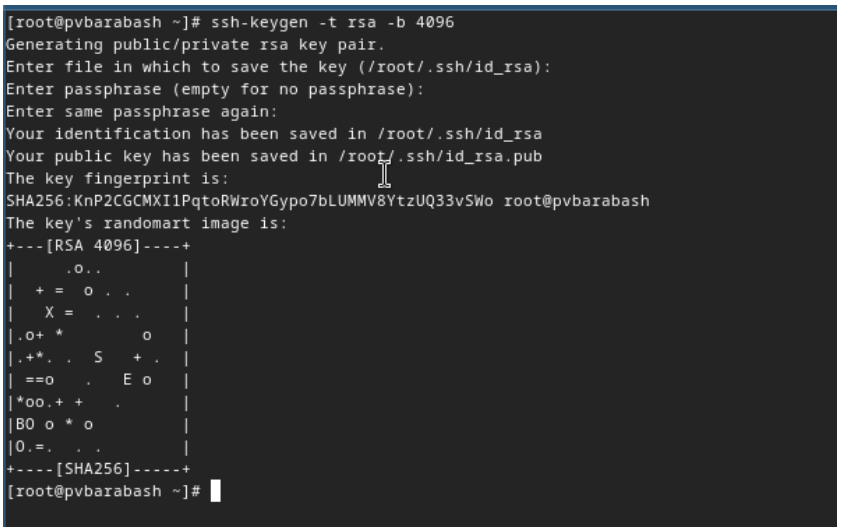

Рис. 3.5: Создание ключа ssh по алгоритму rsa

Я создала ключ по алгоритму ed25519 с помощью команды ssh-keygen -t ed25519 (рис. [3.6]).

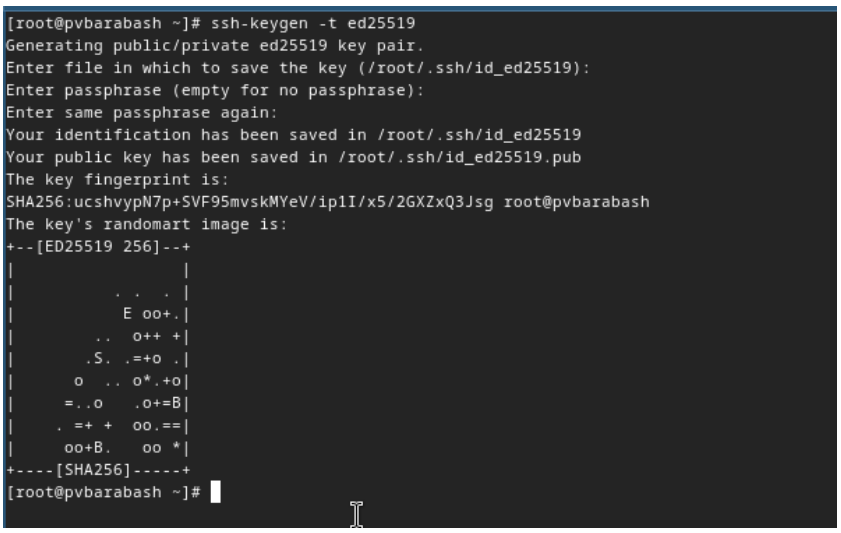

Рис. 3.6: Создание ключа по алгоритму ed25519

Задание 5. Создать ключи PGP.

Я Сгенерировала ключ PGP с помощью команды gpg –full-generate-key. Из предложенных опций выбирала:

тип RSA and RSA;

размер 4096;

срок действия 0 (срок действия не истекает никогда).

Я ввела личную информацию, которая сохранится в ключе: имя и адрес электронной почты (рис. [3.7]).

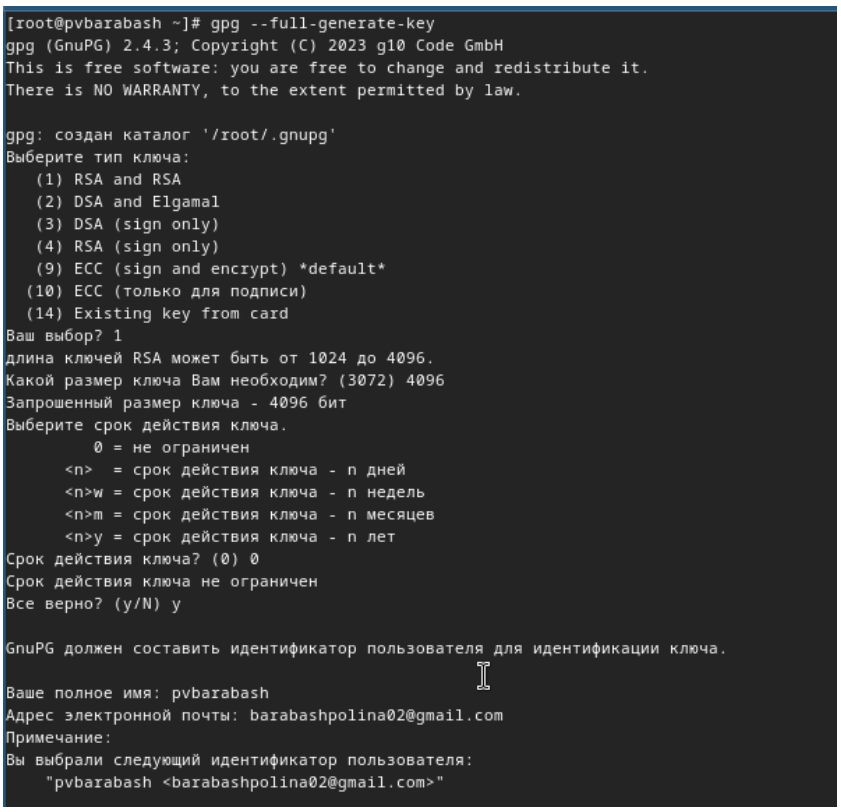

Рис. 3.7: Создание ключа PGP

Задание 6. Добавить PGP ключ в GitHub.

Я вывела список ключей и скопировала отпечаток приватного ключа с помощью gpg –list-secret-keys –keyid-format LONG. Он начинается после после sec (рис. [3.8]).

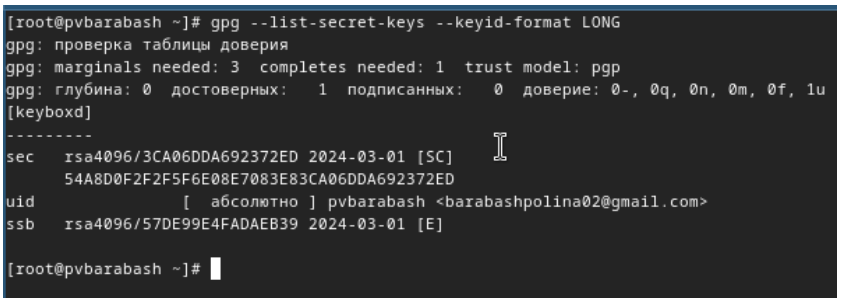

Рис. 3.8: Вывод списка ключей PGP

Для копирования ключа у меня была ошибка с xclip, поэтому я вывела ключ с

помощью команды gpg –armor –export и скопировала его вручную (рис. [3.9]).

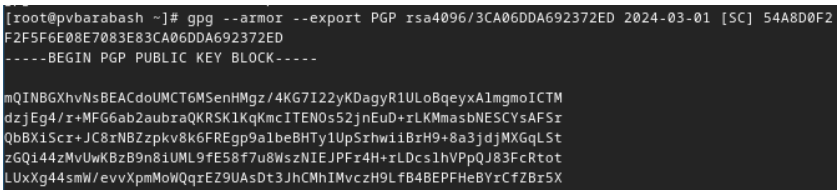

Рис. 3.9: Вывод ключа по отпечатку

Я перешла в настройки GitHub, нажала на кнопку New GPG key и вставила полученный ключ в поле ввода (рис. [3.10]).

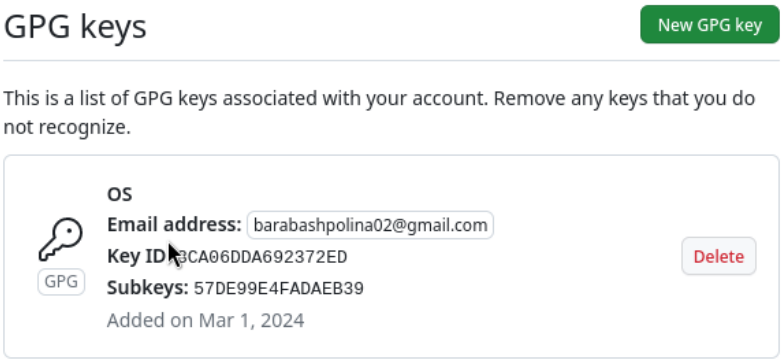

Рис. 3.10: Добавление ключа GPG на GitHub

Задание 7. Используя введёный email, указать Git применять его при подписи коммитов

Я указала Git применять email при подписи коммитов с помощью следующих команд:

git config –global user.signingkey

git config –global commit.gpgsign true

git config –global gpg.program $(which gpg2) (рис. [3.11]).

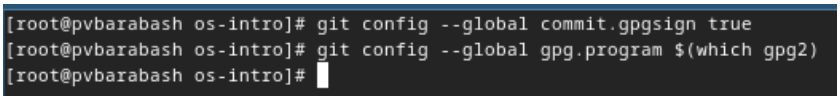

Рис. 3.11: Настройка коммитов

Задание 8. Настроить gh.

Сначала я авторизовалась с помощью команды gh auth login и ответила на наводящие вопросы, а затем авторизовалась через браузер (рис. [3.12]).

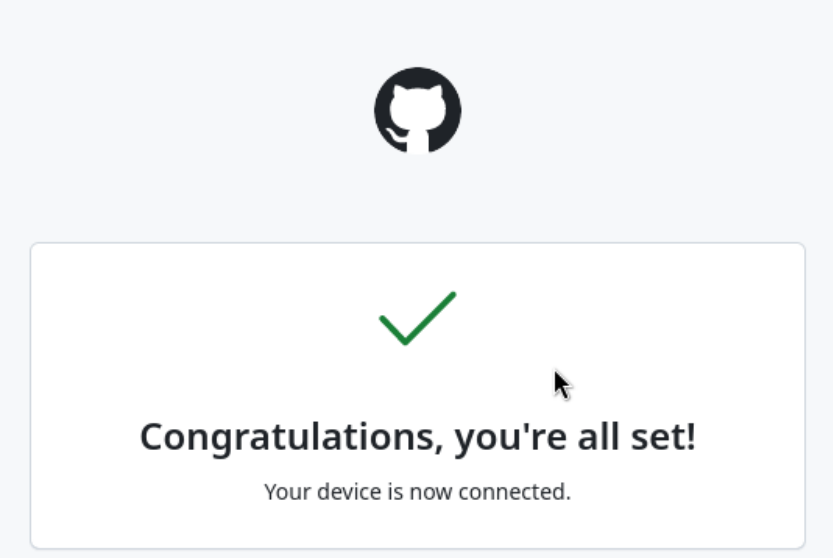

Рис. 3.12: Успешная настройка gh

Задание 9. Создать шаблон рабочего пространства.

Я создала нужный каталог с помощью команды mkdir -p ~/work/study/2022-2023/“Операционные системы”. Перешла в него с помощью команды cd. Так как на github у меня уже был скопирован шаблон рабочего пространства, я клонировала его с помощью команды git clone –recursive git@github.com:/study_2022-2023_os-intro.git os-intro (рис. [3.13]).

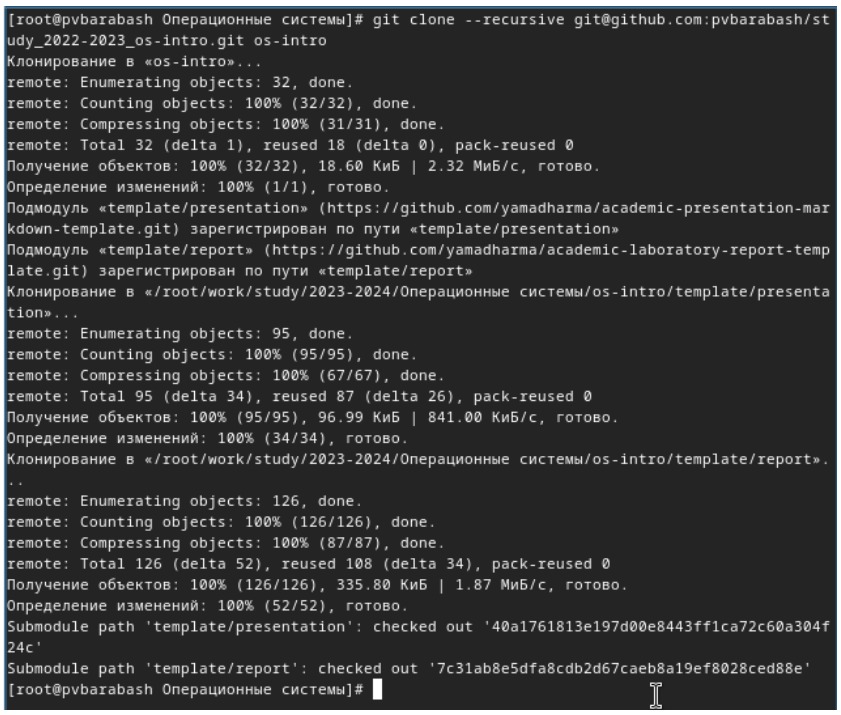

Рис. 3.13: Копирование шаблона рабочего пространства

Задание 10. Настроить каталог курса.

Я перешла в каталог курса с помощью команды cd ~/work/study/2022-2023/“Операционные системы”/os-intro и удалила лишние файлы с помощью команды rm package.json (рис. [3.14]).

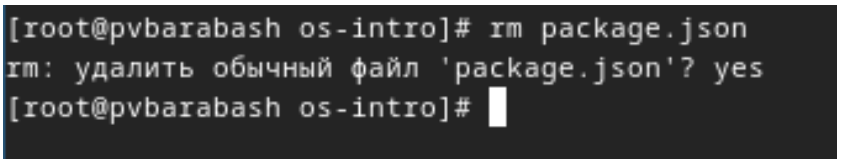

Рис. 3.14: Удаление лишних файлов

Я создала необходимые каталоги с помощью команд echo os-intro > COURSE и make (рис. [3.15]).

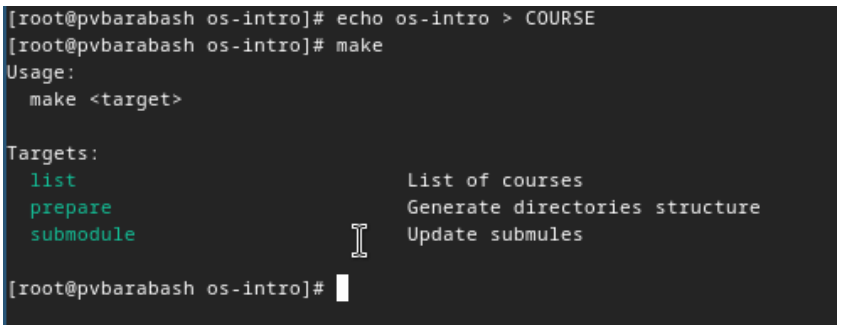

Рис. 3.15: Создание необходимых каталогов

Затем я отправила файлы на сервер (рис. [3.16]).

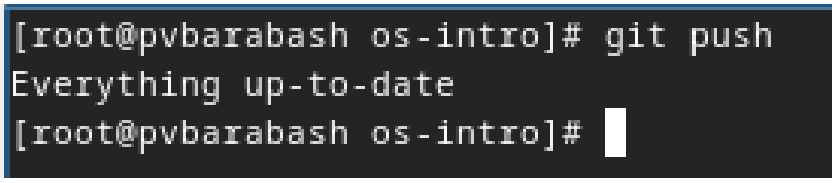

Рис. 3.16: Отправка файлов на сервер

4 Ответы на вопросы

1. **Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?**

Система контроля версий это — программное обеспечение для облегчения работы с изменяющейся информацией. Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом.

2. **Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.**

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. Также он может добавлять коммиты — комментарии о том, что было изменено. Пользователь создает рабочаю копию. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент — система хранит историю версий.

3. **Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.**

В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. То есть классические системы контроля версий являются централизованными. В отличие от классических, в децентрализованных системах контроля версий центральный репозиторий не является обязательным.

Среди классических VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд.

4. **Опишите действия с VCS при единоличной работе с хранилищем.**

При единоличной работе с VCS доступны те же преимущества, что и при совместном использовании (сохранение, откат версий). Отличие в том, что работа с хранилищем доступна только одному пользователю.

5. **Опишите порядок работы с общим хранилищем VCS.**

Для удобства совместной работы с общим хранилищем VCS, стоит добавлять комментарии, что было изменено и подписывать кем были сделаны изменения.

6. **Каковы основные задачи, решаемые инструментальным средством git?**

Основными задачами, решаемыми инструментальным средством git, являются задачи не потерять файлы с исходным кодом, защититься от случайных исправлений и удалений, отменить изменения, если они оказались некорректными, одновременно поддерживать рабочую версию и разработку новой.

7. **Назовите и дайте краткую характеристику командам git.**

Таблица 4.1: Основные команды git

Команда	Значение
git init	Создание основного дерева репозитория
git pull	Получение обновлений (изменений) текущего дерева из центрального репозитория
git push	Отправка всех произведённых изменений локального дерева в центральный репозиторий
git status	Просмотр списка изменённых файлов в текущей директории
git diff	Добавить все изменённые и/или созданные файлы и/или каталоги
git add .	Добавить все изменённые и/или созданные файлы и/или каталоги
git add имена_файлов	Добавить конкретные изменённые и/или созданные файлы и/или каталоги
git rm имена_файлов	Удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории)
git commit -am ‘Описание коммита’	Сохранить все добавленные изменения и все изменённые файлы
git commit	Сохранить добавленные изменения с внесением комментария через встроенный редактор

Команда	Значение
git checkout -b имя_ветки	Создание новой ветки, базирующейся на текущей
git checkout имя_ветки	Переключение на некоторую ветку
git push origin имя_ветки	Отправка изменений конкретной ветки в центральный репозиторий
git merge –no-ff имя_ветки	Слияние ветки с текущим деревом
git branch -d имя_ветки	Удаление локальной уже слитой с основным деревом ветки
git branch -D имя_ветки	Принудительное удаление локальной ветки
git push origin :имя_ветки	Удаление ветки с центрального репозитория

8. **Приведите примеры использования при работе с локальным и удалённым репозиториями.**

Пример использования при работе с удалённым репозиторием:

Работа пользователя со своей веткой начинается с проверки и получения изменений из центрального репозитория (при этом в локальное дерево до начала этой процедуры не должно было вноситься изменений):

git checkout master

git pull

git checkout -b имя_ветки

Пример использования при работе с локальным репозиторием:

Для инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial, необходимо ввести в командной строке:

cd ~

mkdir tutorial

```
cd tutorial
git init
```

9. **Что такое и зачем могут быть нужны ветви (branches)?**

Под веткой принято понимать независимую последовательность коммитов в хронологическом порядке. Однако конкретно в Git реализация ветки выполнена как указатель на последний коммит в рассматриваемой ветке. После создания ветки уже новый указатель ссылается на текущий коммит.

Имя основной ветки Git-проекта по умолчанию — master (однако зачастую бывает main, например, в GitHub), она появляется сразу при инициализации репозитория. Эта ветка ничем не отличается от остальных и также ее можно переименовать, но по договоренности master принято считать главной веткой в проекте.

10. **Как и зачем можно игнорировать некоторые файлы при commit?**

В процессе работы над любым проектом в директории с кодом создаются файлы, которые не являются частью исходного кода. Все эти файлы можно условно разделить на несколько групп:

- Инструментарий:
  - служебные файлы
  - конфигурационные файлы
  - временные файлы редакторов
- Временные файлы:
  - логи — в них содержится полезная информация для отладки, которая собирается во время запуска и работы приложения
  - кеши — файлы, которые нужны для ускорения разных процессов
- Артефакты:

- результаты сборки проекта — например, после компиляции или сборки фронтенда
- зависимости, которые устанавливаются во время разработки — например, node_modules или vendor
- результаты выполнения тестов — например, информация о покрытии кода тестами

Все это в обычной ситуации не должно попадать в репозиторий.

Git позволяет гибко настраивать игнорирование определенных файлов и директорий. Делается это с помощью файла .gitignore, который нужно создать в корне проекта. В этот файл с помощью текстового редактора добавляются имена файлов и директорий, которые надо игнорировать.

5 Выводы

При выполнении данной лабораторной работы я изучила идеологию и применение средств контроля версий и освоила умения по работе с git.