**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ»**

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

**ОТЧЕТ  
по лабораторной работе №2**

по дисциплине «Операционные системы»

Выполнил:

Студент Чванова Ангелина Дмитриевна

Студенческой группы НПИбд-02-21

Студенческий билет № 1032212282

Москва 2022

Лабораторная работа № 2. Управление версиями

Цель работы: изучить идеологию и применение средств контроля версий, а также освоить умения по работе с git.

1.Установка git-flow в Fedora Linux

Установка проходила вручную с помощью данных команд(рис1.1-1.2)

cd /tmp

wget --no-check-certificate -q https://raw.github.com/petervanderdoes ⌋

↪ /gitflow/develop/contrib/gitflow-installer.sh

chmod +x gitflow-installer.sh

sudo ./gitflow-installer.sh install stable

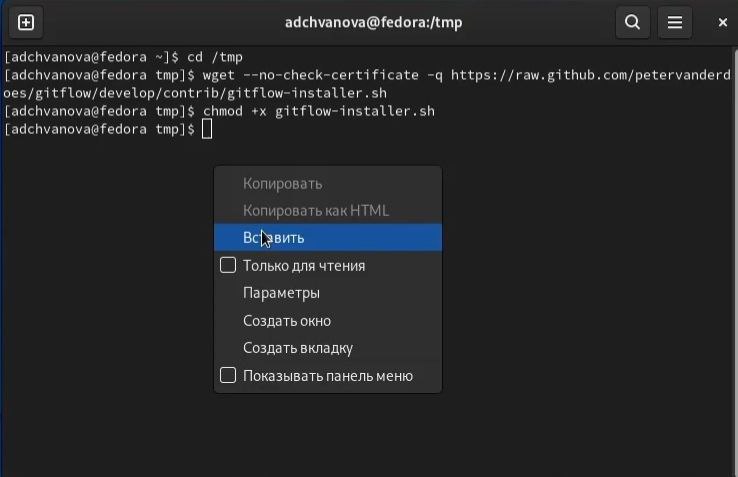


Рис.1.1 терминал с введенными командами для установки git-flow

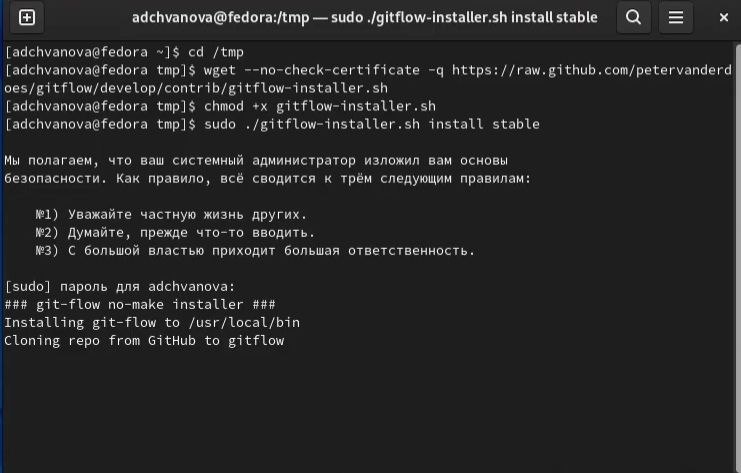


Рис.1.2 терминал с введенными командами для установки git-flow

2.Установка gh в Fedora Linux с помощью команды

sudo dnf install gh (рис2)

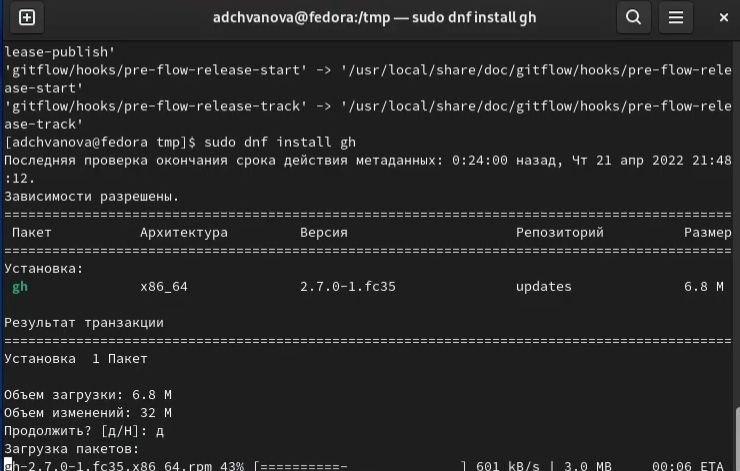


Рис.2 терминал с введенной командой для установки gh

3.Базовая настройка git(рис3.1-3.2)

– Задаем имя и email владельца репозитория с помощью команд:

git config --global user.name "Name Surname"

git config --global user.email "work@mail"

Настройка utf-8 в выводе сообщений git:

git config --global core.quotepath false

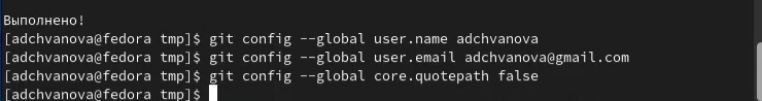


Рис.3.1 Ввод имени владельца и email , а также настройка utf-8

Настройте верификацию и подписание коммитов git.

– Зададаем имя начальной ветки (будем называть её master):

git config --global init.defaultBranch master

– Параметр autocrlf:

git config --global core.autocrlf input

– Параметр safecrlf:

git config --global core.safecrlf warn

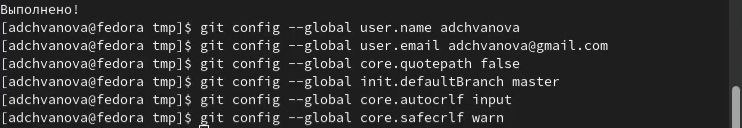


Рис.3.2 Выбор имени начальной ветки и настройка параметров safecrlf и autocrlf

4.Создание ключей ssh(рис4)

Ключ создавался по алгоритму rsa с размером для ключа 4096 бит:

ssh-keygen -t rsa -b 4096

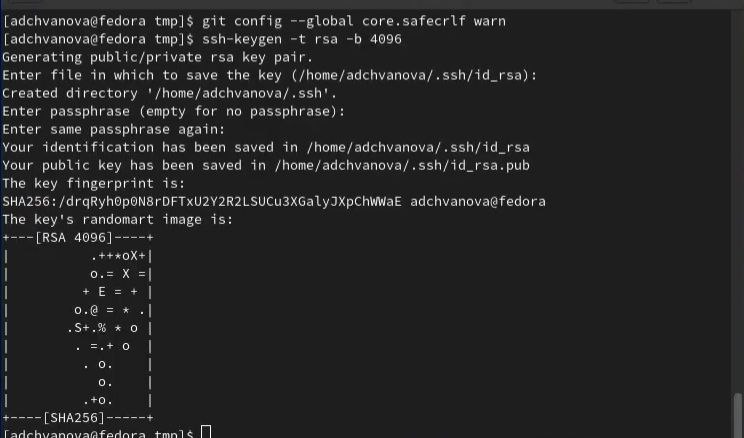


Рис.4 Создание ключа ssh по алгоритму rsa с размером для ключа 4096 бит в консоле

5.Создание ключей pgp(рис5.1-5.2)

– Генерируем ключ

gpg --full-generate-key

– Из предложенных опций (рис5.1) выбираем:

– тип RSA and RSA;

– размер 4096;

– выберите срок действия; значение по умолчанию— 0 (срок действия не истекает

никогда).

– GPG запросит личную информацию, которая сохранится в ключе:

– Имя (не менее 5 символов).

– Адрес электронной почты.

– При вводе email убедитесь, что он соответствует адресу, используемому на

GitHub.

– Комментарий.

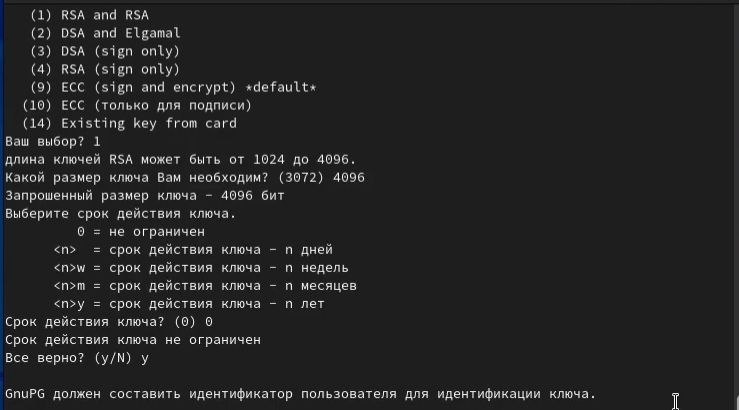


Рис.5.1 консоль с введенной командой для создания ключей pgp и предложенные опции

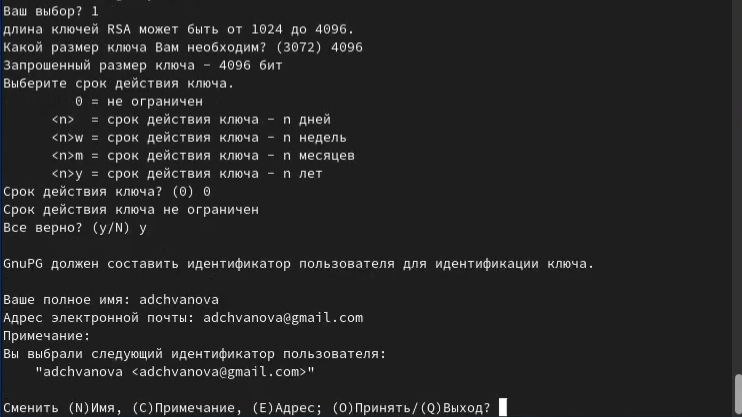


Рис.5.2 выбранные опции для создания ключей pgp

6.Добавление PGP ключа в GitHub(рис6.1-6.3)

– Выводим список ключей и копируем отпечаток приватного ключа:

gpg --list-secret-keys --keyid-format LONG

– Формат строки (рис6.1):

sec Алгоритм/Отпечаток\_ключа Дата\_создания [Флаги] [Годен\_до] ID\_ключа

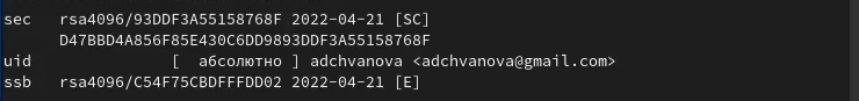
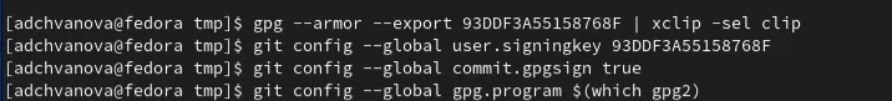


Рис.6.1 Сгенерированный в консоли ключ

– Копирование сгенерированного PGP ключа в буфер обмена(рис6.2):

1 gpg --armor --export <PGP Fingerprint> | xclip -sel clip

Рис.6.2 Копирование в консоле сгенерированного PGP ключа в буфер обмена

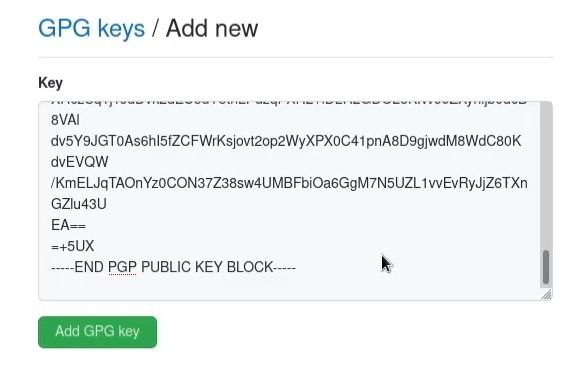


Рис.6.3 вставка полученного ключа в поле ввода (переход в настройки GitHub (https://github.com/settings/keys)

7.Настройка автоматических подписей коммитов git (рис7)

– Используя введёный email, указываем Git применять его при подписи коммитов:

git config --global user.signingkey <PGP Fingerprint>

git config --global commit.gpgsign true

git config --global gpg.program $(which gpg2)

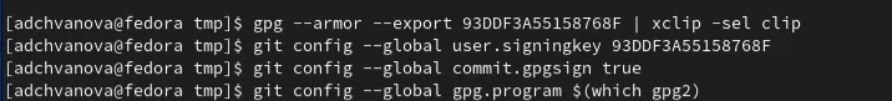


Рис.7 терминал с введенными командами для настройки автоматических подписей коммитов git

8.Настройка gh(рис8)

–команда для авторизации

gh auth login

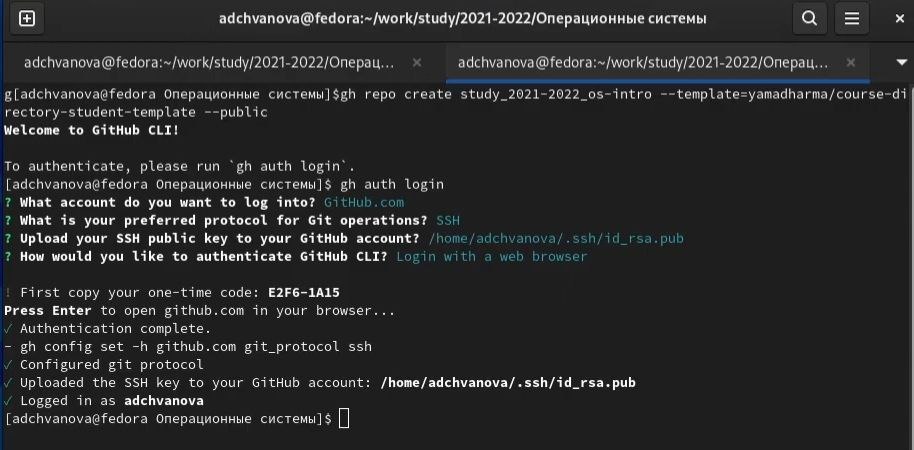


Рис.8 Авторизация в gh

9.Сознание репозитория курса на основе шаблона и настройка каталога курса (рис9.1-9.2)

–создание шаблона рабочего пространства.

mkdir -p ~/work/study/2021-2022/"Операционные системы"

cd ~/work/study/2021-2022/"Операционные системы"

gh repo create study\_2021-2022\_os-intro

↪ --template=yamadharma/course-directory-student-template --public

git clone --recursive

↪ git@github.com:<owner>/study\_2021-2022\_os-intro.git os-intro

– Переход в каталог курса:

cd ~/work/study/2021-2022/"Операционные системы"/os-intro

– Удаление лишних файлов(рис9.1):

rm package.json

– Создайте необходимые каталоги(рис9.1):

make COURSE=os-intro

– Отправьте файлы на сервер(рис9.1):

git add .

git commit -am 'feat(main): make course structure'

git push

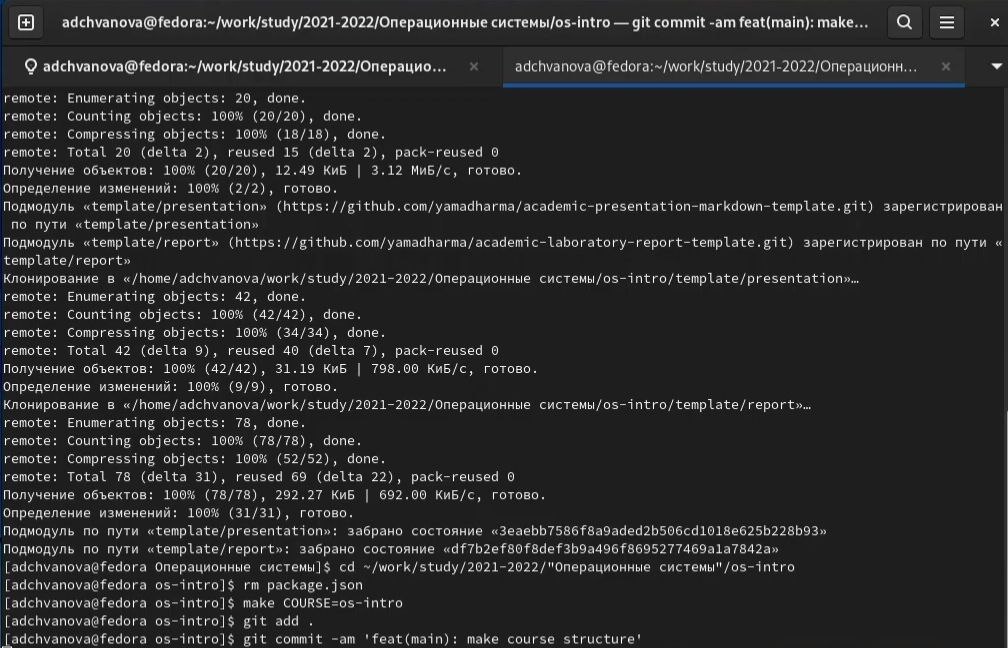


Рис.9.1 удаление лишних файлов, создание каталогов и отправка файлов на сервер

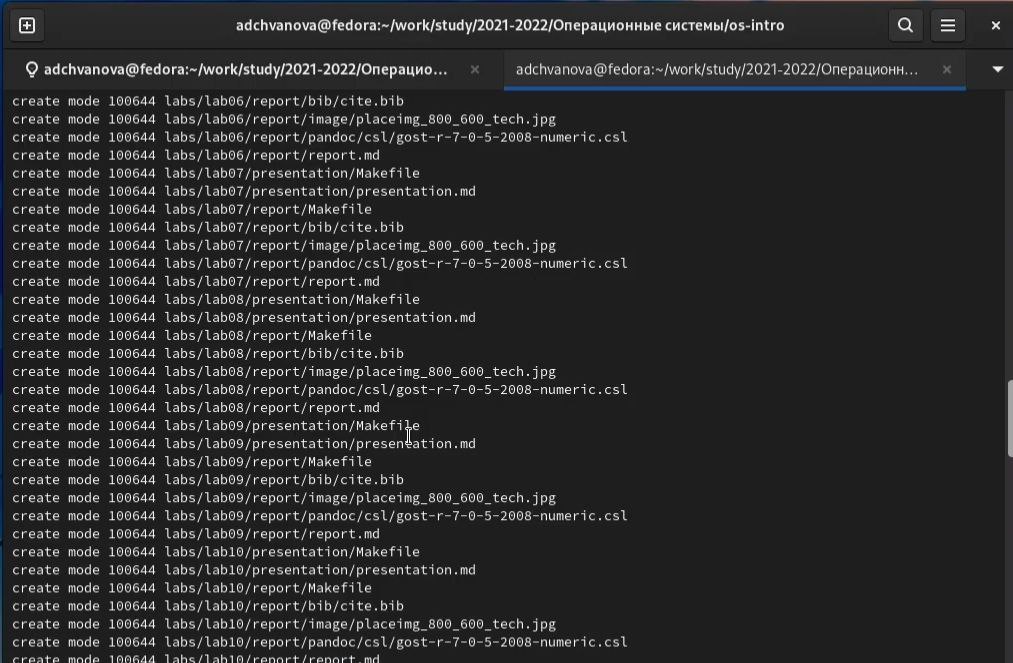


Рис9.2 отправка файлов на сервер

Вывод: были изучены некоторые команды в консоли для работы с GitHub и применены средства контроля версий, а также освоены умения по работе с git. Удалось создать репозиторий курса на основе шаблона и настроить его. Файлы были отправлены на сервер.

Контрольные вопросы

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

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.

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

Хранилище – место хранения 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.

Commit позволяется отправлять изменения на сервер VCS.

История — список всех изменений проекта с возможностью отката в любую точку истории.

Рабочая копия (working copy) — текущее состояние файлов проекта (любой версии), полученных из хранилища и, возможно, измененных.

1. Что представляют собой и чем отличаются централизованные и децентрализованныеVCS? Приведите примеры VCS каждого вида.

В децентрализованных системах каждый узел принимает свое собственное решение. Конечное поведение системы является совокупностью решений отдельных узлов. Обратите внимание, что нет единого объекта, который получает и отвечает на запрос.

Bitcoin. Давайте возьмем биткойны, например, потому что это самый популярный пример использования децентрализованных систем. Ни одна организация / организация не владеет сетью биткойнов. Сеть представляет собой сумму всех узлов, которые общаются друг с другом для поддержания количества биткойнов, которое есть у каждого владельца счета.

Централизованные системы — это системы, которые используют архитектуру клиент / сервер, где один или несколько клиентских узлов напрямую подключены к центральному серверу. Это наиболее часто используемый тип систем во многих организациях, где клиент отправляет запрос на сервер компании и получает ответ.

Wikipedia. Рассмотрим огромный сервер, на который мы отправляем наши запросы, и сервер отвечает запрашиваемой статьей. Предположим, мы ввели поисковый запрос «нездоровая пища» в строке поиска Википедии. Этот поисковый запрос отправляется как запрос на серверы Википедии (в основном, расположенные в штате Вирджиния, США), которые затем возвращают статьи, основанные на релевантности. В этой ситуации мы являемся клиентским узлом, серверы Википедии являются центральным сервером.

1. Опишите действия с VCS при единоличной работе с хранилищем.

Создадим локальный репозиторий. Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория:

git config —global user.name"Имя Фамилия"

git config —global user.email"work@mail"

и настроив utf-8 в выводе сообщенийgit:

git config —global quotepath false

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

cd

mkdir tutorial

cd tutorial

git init

5. Опишите порядок работы с общим хранилищем VCS.

Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый):

ssh-keygen -C"Имя Фамилия <work@mail>"

Ключи сохраняться в каталоге~/.ssh/.

Скопировав из локальной консоли ключ в буфер обмена

cat ~/.ssh/id\_rsa.pub | xclip -sel clip

вставляем ключ в появившееся на сайте поле.

6. Каковы основные задачи, решаемые инструментальным средством git?

У Git две основных задачи: первая — хранить информацию о всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом. А также ведение истории изменений, фиксирование изменений, совмещение версий, веток, откат к прошлым версиям.

7. Назовите и дайте краткую характеристику командам git.

git init - инициализирует локальный репозиторий

git add \* или add. - добавляет файлы в репозиторий

git commit - версия фиксации

git pull - загружает текущую версию проекта

git push - отправляет измененный проект на сервер

git checkout - позволяет переключаться между ветками

git status - текущий статус проекта

git branch - просмотреть доступные ветки

git remote add - добавить удаленный репозиторий

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

Использование git с локальными репозиториями используется для добавления, например, текстового файла в локальный репозиторий

git add file.txt

git commit –am ФАЙЛ

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

Ветви необходимы, чтобы иметь возможность "разделять" части работы и работать отдельно над каждой имплементацией. Использование ветвей дает возможность обрабатывать нововведения в основную ветвь, которая чаще всего является релизной.

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

Игнорировать файлы при commit можно с помощью .gitignore файла.

В нем указываются пути, названия, расширения и другие идентификации

нежелательных объектов которые не будут учитываться в commit. Это

полезно для исключения как "мусорных" файлов, которые не являются

значимой частью проекта, а также конфиденциальных файлов, которые

содержат в себе приватную информацию, такую как пароли и токены.