Отчёт по лабораторной работе №2

Дисциплина: Операционные системы

Батова Ирина Сергеевна, НММбд-01-22

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Целью данной лабораторной работы является изучение идеологии и применение средств контроля версий, а также освоение умения по работе с git.

2 Выполнение лабораторной работы

Первым делом устанавливаем git (рис. 2.1).

```
[isbatova@fedora ~]$ dnf install git
```

Рис. 2.1: Установка git

Далее устанавливаем gh (рис. 2.2).

```
[isbatova@fedora ~]$ dnf install gh
```

Рис. 2.2: Установка gh

После этого приступаем к базовой настройке git (рис. 2.3). Для этого задаем имя и email владельца репозитория (строки 1 и 2), далее настраиваем utf-8 в выводе сообщений git (строка 3), а также задаем имя начальной ветки (строка 4), параметр autorclf (строка 5) и параметр saferclf (строка 6).

```
[isbatova@fedora ~]$ git config --global user.name "<Irina Batova>"
[isbatova@fedora ~]$ git config --global user.email "<1132226490@pfur.ru>"
[isbatova@fedora ~]$ git config --global core.quotepath false
[isbatova@fedora ~]$ git config --global init.defaultBranch master
[isbatova@fedora ~]$ git config --global core.autocrlf input
[isbatova@fedora ~]$ git config --global core.safecrlf warn
```

Рис. 2.3: Базовая настройка git

После этого необходимо создать ssh ключ. Для этого используется команда “ssh-keygen -C” (рис. 2.4).

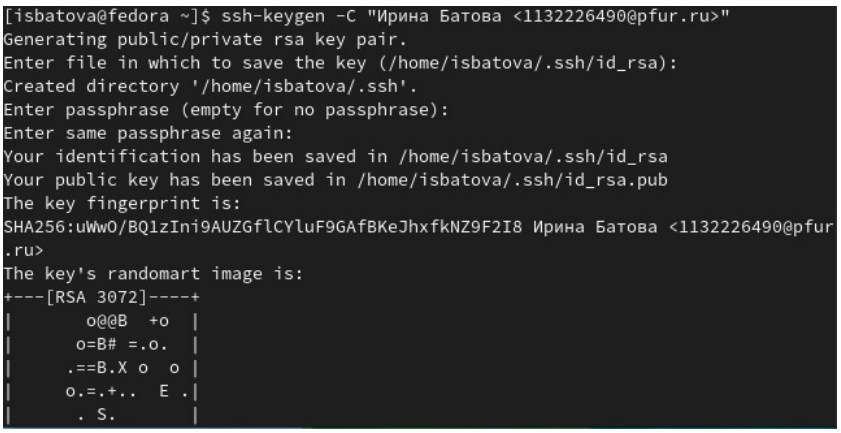

Рис. 2.4: Генерация ssh ключа

После этого нам необходимо загрузить этот ключ. Для этого заходим на сайт github.com, переходим в Setting – SSH and GPG keys – New SSH key (рис. 2.5).

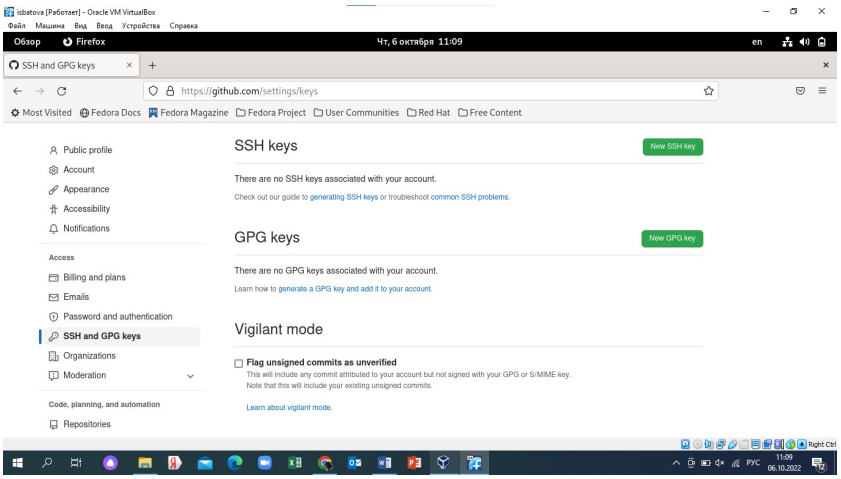

Рис. 2.5: Создание ssh ключа на github

Далее возвращаемся в терминал и вводим команду “cat ~/.ssh/id_rsa.pub | xclip -sel clip”, чтобы скопировать ключ. В процессе соглашаемся на установление пакета “xclip” (рис. 2.6).

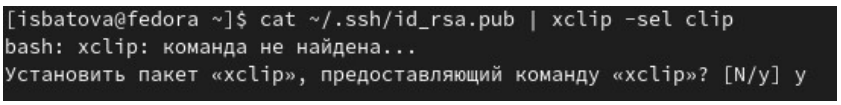

Рис. 2.6: Копирование ssh ключа

После этого возвращаемся на github.com, вводим название ключа “Laptop_home” и в поле “Key” вставляем ключ (рис. 2.7). Ключ создан (рис. 2.8).

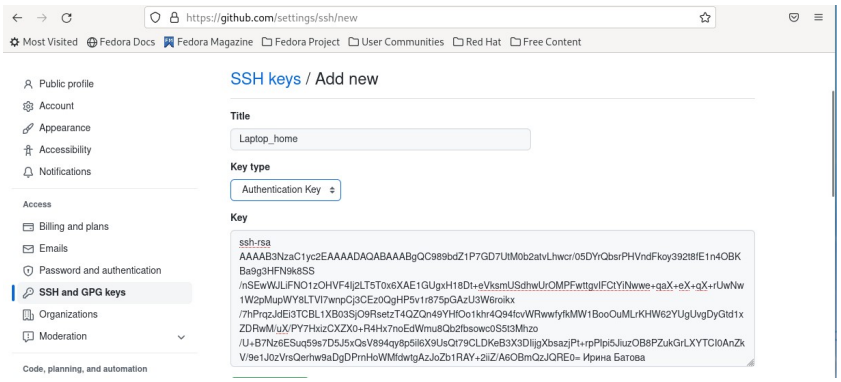

Рис. 2.7: Добавление ssh ключа на github

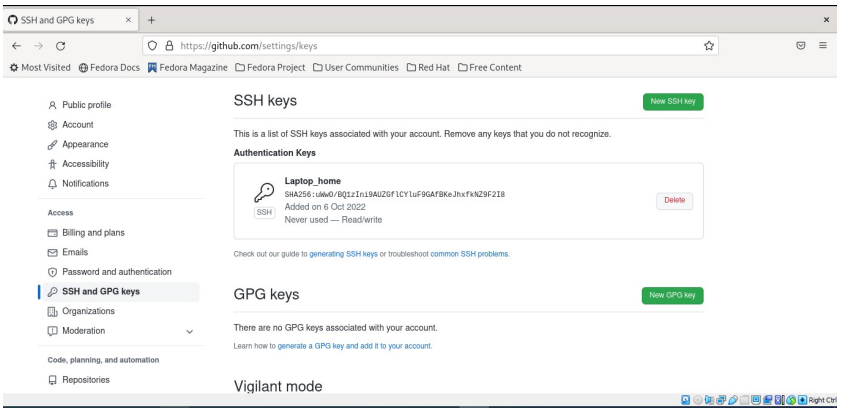

Рис. 2.8: Ключ создан

После создания ssh ключа нам также нужно создать gpg ключ. Для этого заходим в терминал и вводим команду ‘gpg –full-generate-key’. Нам будут предлагаться разные опции. Тип ключа выбираем ‘RSA and RSA’ (рис. 2.9).

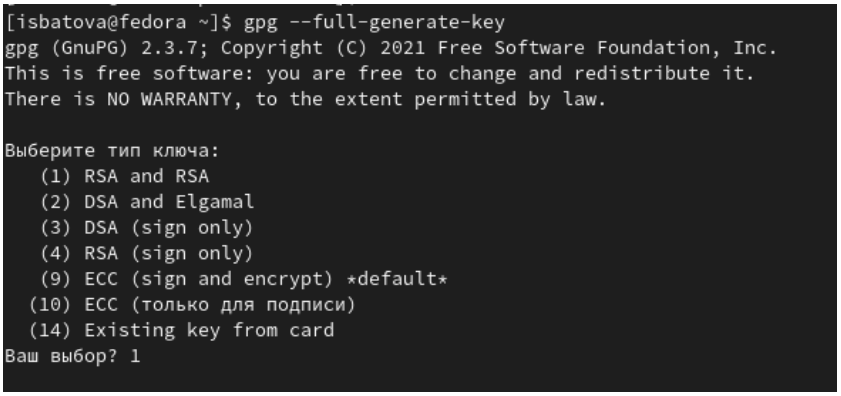

Рис. 2.9: Выбор типа gpg ключа

Следующим шагом выбираем размер (4096 бит) и срок действия ключа - неограниченный (рис. 2.10).

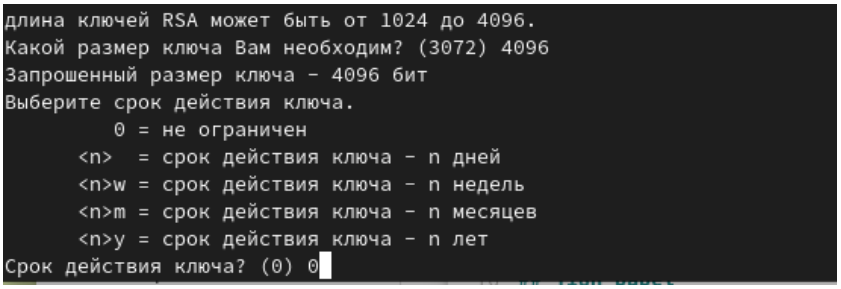

Рис. 2.10: Выбор размера и срока действия gpg ключа

Далее вводим имя и email (рис. 2.11). Примечание оставляем пустым.

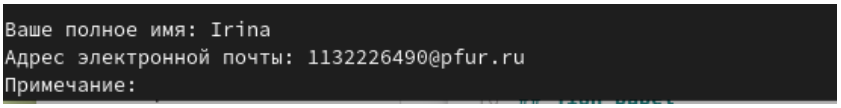

Рис. 2.11: Ввод имени и почты

Далее задаем фразу-пароль (рис. 2.12). Ключ сгенерирован (рис. 2.13).

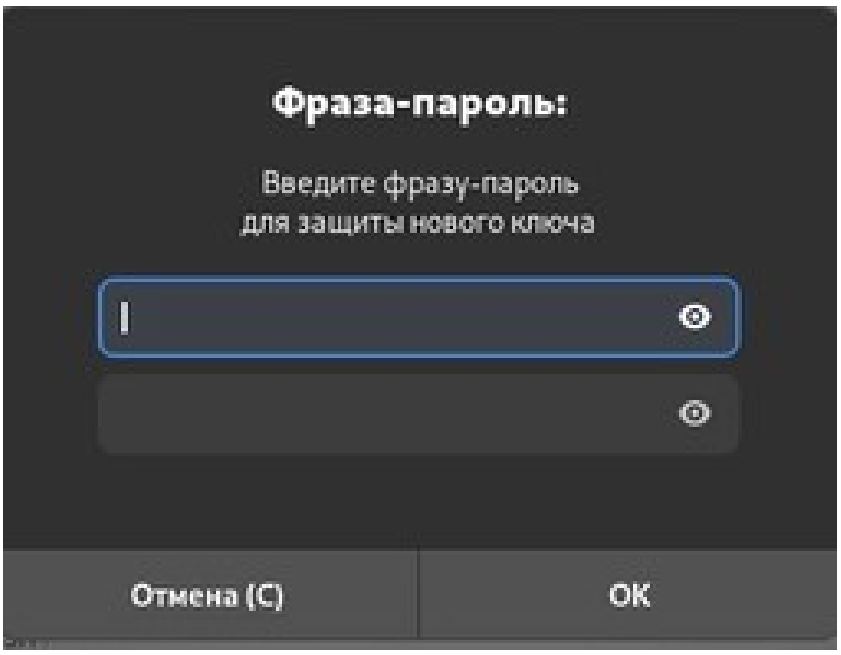

Рис. 2.12: Задание пароля

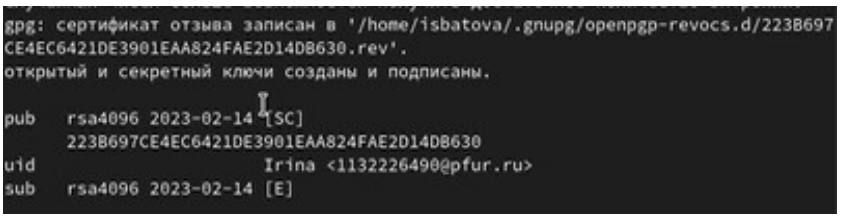

Рис. 2.13: gpg ключ сгенерирован

После этого выводим список ключей командой ‘gpg –list-secret-keys –keyid-format LONG’ (рис. 2.14) и находим отпечаток нужного нам ключа.

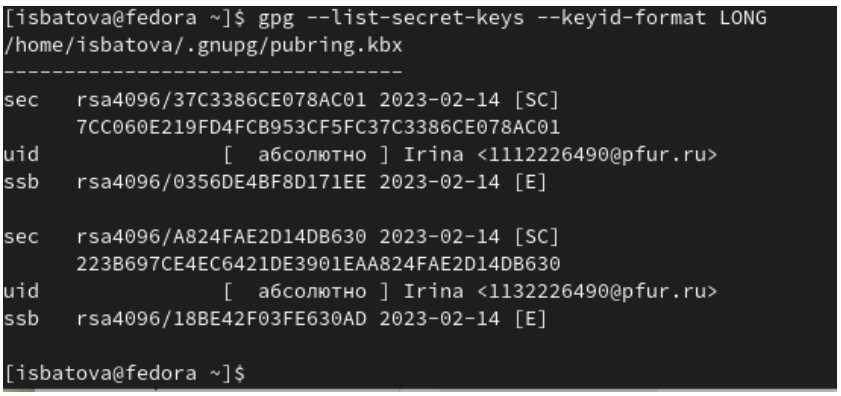

Рис. 2.14: Вывод списка ключей

С помощью отпечатка и команды ‘gpg –armor –export PGP Fingerprint | xclip -sel clip’ копируем ключ в буфер обмена (рис. 2.15).

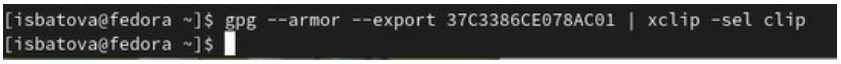

Рис. 2.15: Копирование gpg ключа в буфер обмена

После этого заходим на github и загружаем на ключ туда (рис. 2.16).

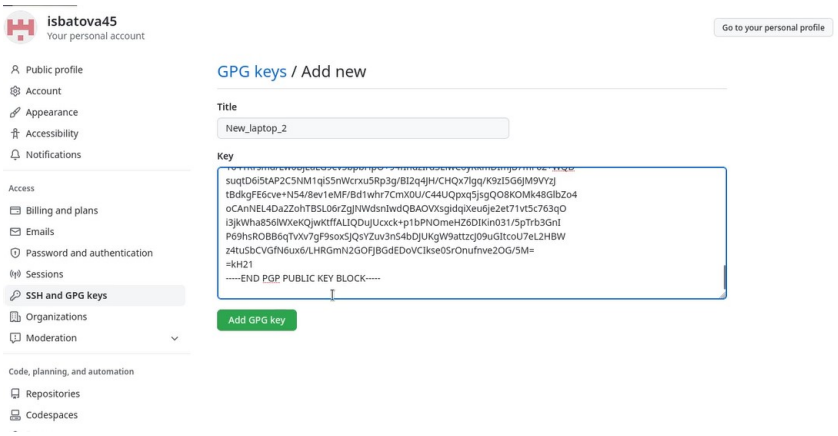

Рис. 2.16: Добавление pgp ключа на github

Следующим нашим шагом будет настройка автоматических подписей коммитов git (рис. 2.17).

Рис. 2.17: Настройка автоматических подписей коммитов

Далее настраиваем gh. Для этого вводим команду ‘gh auth login’ и отвечаем на наводящие вопросы (рис. 2.18).

Рис. 2.18: Настройка gh

Позже нас перекидывает в браузер, где мы вводим код и успешно заканчиваем настройку gh (рис. 2.19).

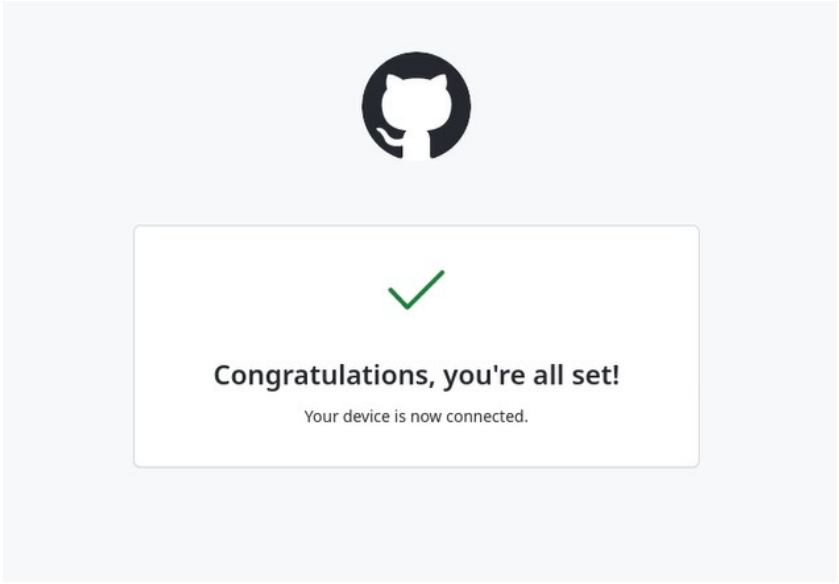

Рис. 2.19: Завершение настройки gh

Далее нам необходимо создать репозиторий курса и настроить каталог курса.

Для начала с помощью команды ‘mkdir -p’ создаем необходимые каталоги и с помощью команды ‘cd’ переходим в него (рис. 2.20).

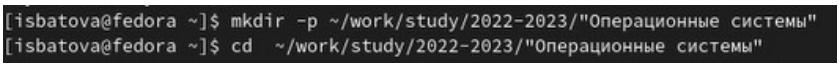

Рис. 2.20: Создание каталогов курса

Далее создаем репозиторий на github (рис. 2.21) и клонируем его (рис. 2.22).

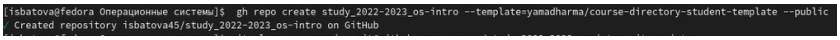

Рис. 2.21: Создание репозитория

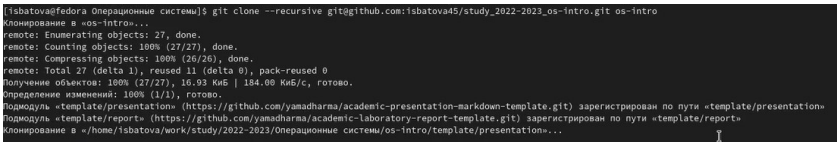

Рис. 2.22: Клонирование репозитория

Продолжаем настройку каталога курса. Переходим в каталог курса (рис. 2.23).

Рис. 2.23: Переход в каталог курса

Далее удаляем лишние файлы и создаем необходимые каталоги (рис. 2.24).

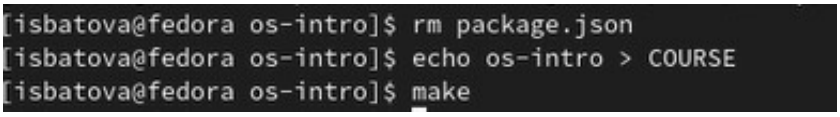

Рис. 2.24: Удаление лишних файлов и создание каталогов

Последним шагом отправляем файлы на сервер (рис. 2.25, рис. 2.27). Для отправки вводим фразу-пароль (рис. 2.26).

Рис. 2.25: Отправление файлов на сервер 1

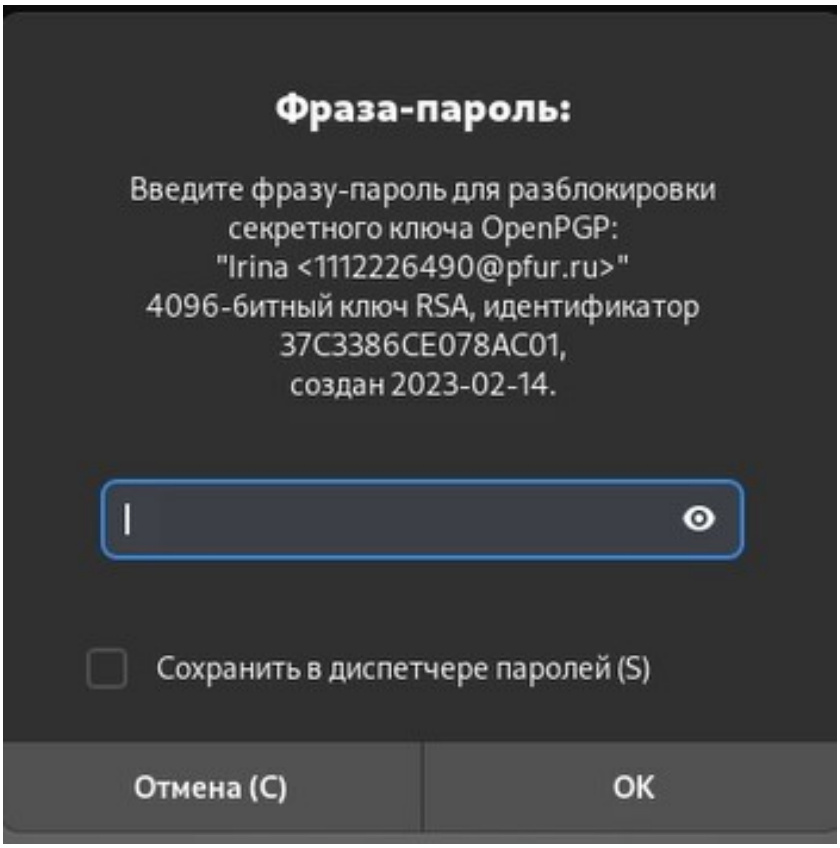

Рис. 2.26: Ввод фразы-пароля

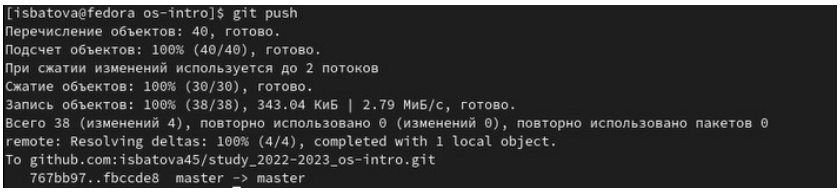

Рис. 2.27: Отправление файлов на сервер 1

3 Выводы

В данной лабораторной работе мной были изучены идеологии и применение средств контроля версий, а также освоены умения по работе с git.

4 Контрольные вопросы

1. Система контроля версий (VCS) - это система, регистрирующая изменения в файлах. VCS может хранить несколько версий одного документа, позволяет возвращаться к более ранним версиям, показывает, кто и какие конкретно изменения внес в документ. Система контроля версий обычно применяется в случае, если над одним проектом работает группа людей.

2. При выполнении участником проекта своей части работы он получает нужную ему версию файлов из хранилища, а затем сохраняет новую версию файлов в хранилище. То есть централизованное хранилище сохраняет все файлы - и до правки участником, и после. Следующий пользователь перед работой начнет работу с новой, измененной версией файла. Команда commit как раз осуществляет сохранение изменений (но при этом на сервер они уходят с помощью другой команды). История проекта - это история изменения файлов, то есть кто изменил, какие файлы, какие изменения были внесены. Рабочую версию участник проекта всегда извлекает перед началом работы - то есть, рабочей версией называется та, в которой сохранены все необходимые для работы конкретного пользователя изменения.

3. Централизованные VCS созданы для бэкапирования, отслеживания и синхронизации файлов. Все изменения происходят через центральный сервер. А в децентрализованных VCS у каждого пользователя есть свой полноценный репозиторий и нет жестко заданной структуры репозиториев с центральным сервером. Децентрализованные VCS были созданы для обмена

изменениями, например, Git. Пример централизованной VCS - Subversion (SVN).

4. При единоличной работе с хранилищем для начала создаем локальный репозиторий и делаем предварительную конфигурацию: задаем имя и email владельца репозитория, настраиваем utf-8 в выводе сообщений, задаем имя начальной ветки, параметр autorclf и параметр saferclf. Далее при внесении изменений в файлы необходимо вводить последовательность команд ‘git add .’, ‘git commit -am’, ‘git push’ для сохранении изменений в репозитории.

5. При работе с общим хранилищем VCS нам необходим ssh ключ. Для этого используется команда “ssh-keygen -C”, после этого копируем ключ и вставляем в соотвествующее окно в веб-браузере

6. У git можно выделить следующие основные задачи:

- хранение всей информации о любых изменениях в файлах
- фиксирование и совмещение изменений
- сохранение истории и возможность вернуться к ранней версии файла

7. Основные команды git:

- 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 - просмотр текущих изменений
- git add . - добавление всех изменённых и/или созданных файлов и/или каталогов

- git add имена_файлов - добавление конкретных изменённых и/или созданных файлов и/или каталогов
- git rm имена_файлов - удаление файла и/или каталога из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории)
- git commit -am ‘Описание коммита’ - сохранение всех добавленных изменений и всех изменённых файлов
- git commit - сохранение добавленных изменений с внесением комментария через встроенный редактор
- 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. Например, добавление файла ‘text.txt’

- git add text.txt
- git commit -am ‘add file text.txt’
- git push

9. Ветка в git - это указатель на один из коммитов. Ветка берет свое начало от какого-то из коммитов. Ветки существуют для того, чтобы пользователи

могли работать над проектом, не мешая друг другу. Обычно при работе участник проекта заводит новую ветку от последнего рабочего коммита базовой ветки, а после решения задачи объединяет созданную ветку с базовой.

10. При работе над проектом могут создаваться файлы, которые не нуждаются в выгрузке в репозиторий. Игнорируемые файлы отслеживаются в специальном файле .gitignore - для игнорирования каких-то файлов достаточно их здесь указать.