РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятности

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

дисциплина: Операционные системы

Студент: Агаджанян Артур Вячеславович Группа: НКАбд-01-23

Москва

2024 г.

Управление версиями

Цель работы:

Изучить идеалогию и примение средств котроля версий. Освоить умения по работе с git.

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.

Основные команды git

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 .

- добавить конкретные изменённые и/или созданные файлы и/или каталоги:

git add имена_файлов

- удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории):

git rm имена_файлов

Сохранение добавленных изменений:

- сохранить все добавленные изменения и все изменённые файлы:

git commit -am 'Описание коммита'

- сохранить добавленные изменения с внесением комментария через встроенный редактор:

git commit

- создание новой ветки, базирующейся на текущей:

git checkout -b имя_ветки

- переключение на некоторую ветку:

git checkout имя_ветки

- (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) отправка изменений конкретной ветки в центральный репозиторий:

git push origin имя_ветки

- слияние ветки с текущим деревом:

git merge --no-ff имя_ветки

Удаление ветки:

- удаление локальной уже слитой с основным деревом ветки:

git branch -d имя_ветки

- принудительное удаление локальной ветки:

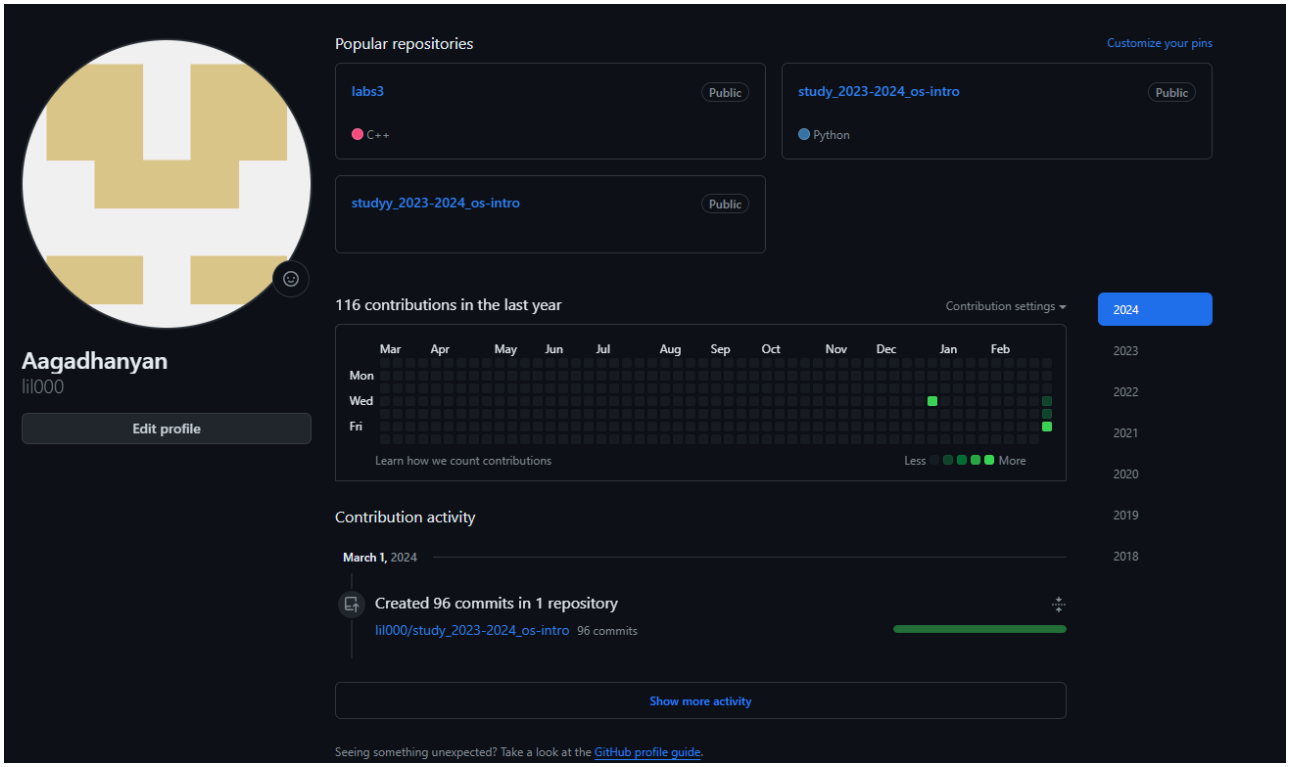
git branch -D имя_ветки

- удаление ветки с центрального репозитория:

git push origin :имя_ветки

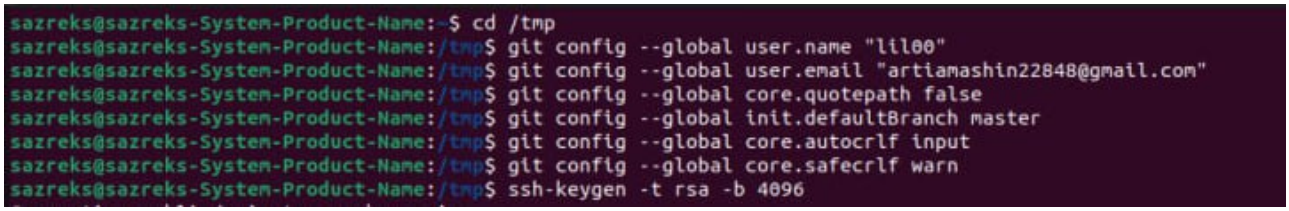
Ход работы

1. Создайте учетную запись на https://github.com/ (на скриншоте она уже создана)



2. Установка программного обеспечения(оно уже было установлено,снизу приведены команды,которые я использовал) ![Установка]
   dnf install gh

3. Базовая настройка git



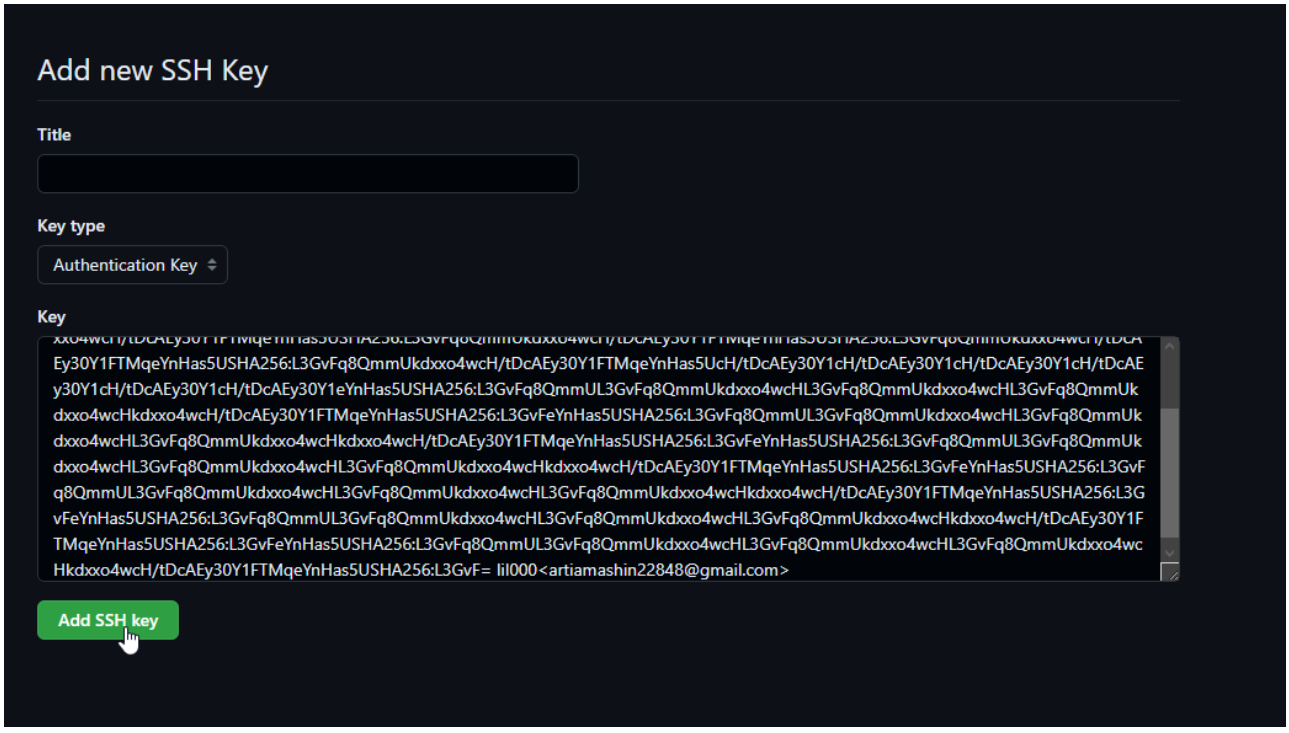
4. Создаем ключ SSH

```
Enter file in which to save the key (/home/sazreks/.ssh/id_rsa):
/home/sazreks/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/sazreks/.ssh/id_rsa
Your public key has been saved in /home/sazreks/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:HRqpT0h72+qa7566wOBECM8U9XrVgE8YHlM+aGFjWGY sazreks@sazreks-System-Product-Name
The key's randomart image is:
+---[RSA 4096]----+
|. oo.oE=o        |
|.=  .BoB.+       |
|. +   *o* o      |
| .   + =.= .     |
|  o . = S .      |
| o o . + o       |
|  . o   o .      |
|     . . o       |
|      =OB        |
+----[SHA256]-----+
```

Скриншот: Устанавливаем ключ по алгоритму rsa с ключем размером **4096 бит**

```
sazreks@sazreks-System-Product-Name:/tmp$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/sazreks/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/sazreks/.ssh/id_ed25519
Your public key has been saved in /home/sazreks/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:XpemMk5VRlCN18CC9YyLfqSZRnYhlcslrhKIPVvz0As sazreks@sazreks-System-Product-Name
The key's randomart image is:
+--[ED25519 256]--+
|          .===.o |
|          .o=+= .|
|     o . ..o=*o  |
|    . + E .=++   |
|       +S**oB    |
|      ...*+X     |
|        =.B .    |
|       o + .     |
|        .        |
+----[SHA256]-----+
```

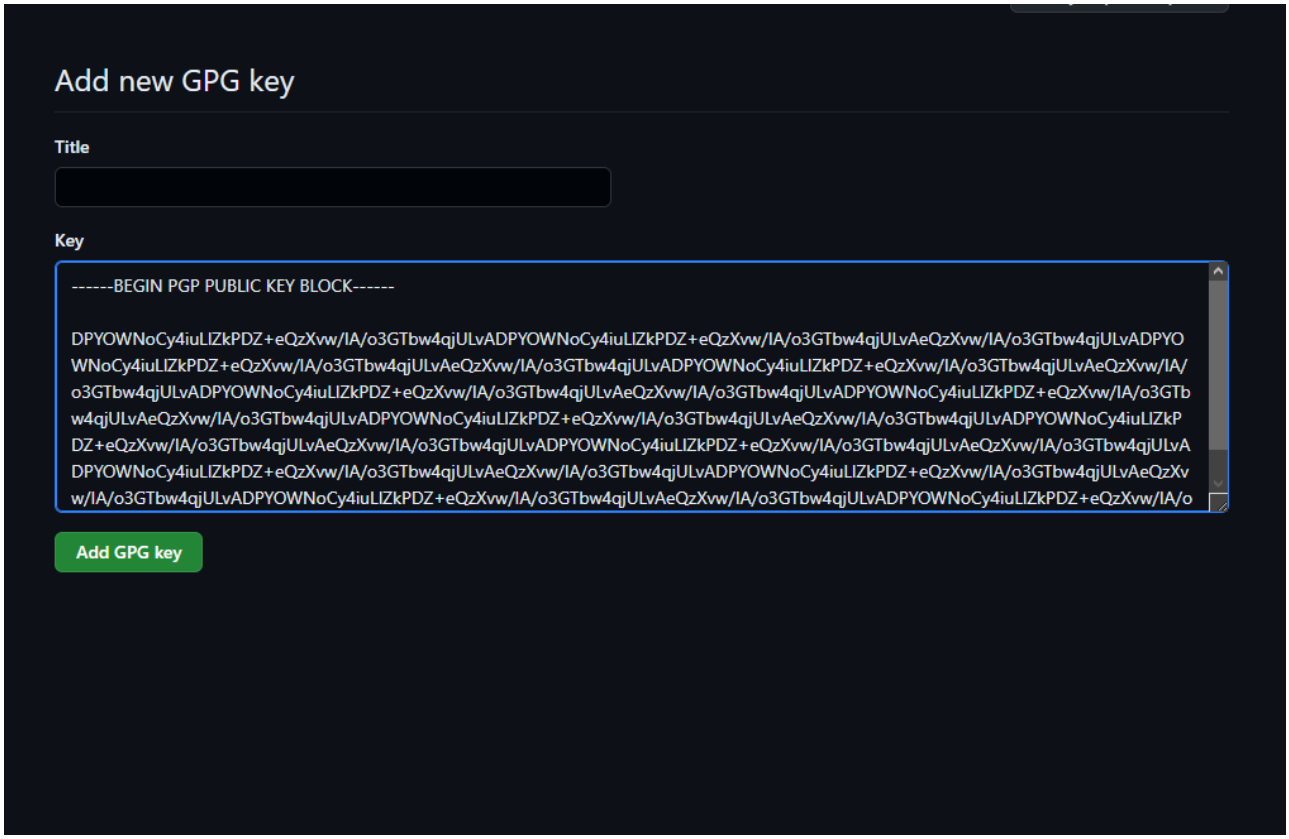
Скриншот: Ключ алгоритма **ed25519** Добавляем наш SHH ключ в гитхаб



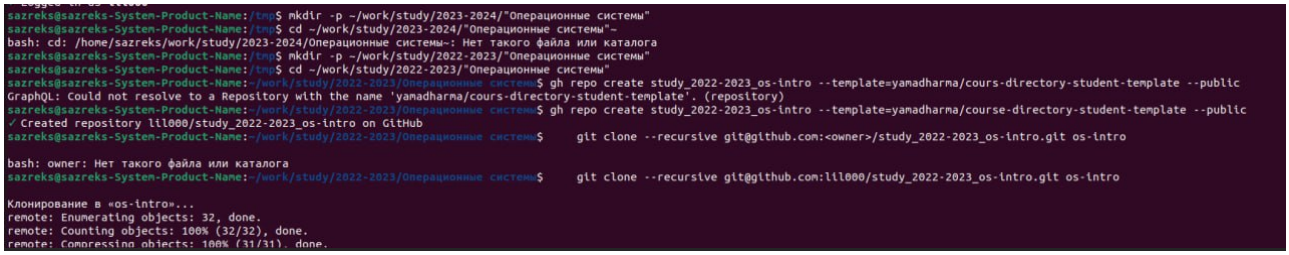
5. Создаем ключ PGP

```
sazreks@sazreks-System-Product-Name:/tmp$ gpg--list-secret-keys --keyid-format LONG
gpg--list-secret-keys: команда не найдена
sazreks@sazreks-System-Product-Name:/tmp$ gpg --list-secret-keys --keyid-format LONG
sazreks@sazreks-System-Product-Name:/tmp$ gpg --full-generate-key
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Выберите тип ключа:
   (1) RSA и RSA (по умолчанию)
   (2) DSA и Elgamal
   (3) DSA (только для подписи)
   (4) RSA (только для подписи)
  (14) Имеющийся на карте ключ
Ваш выбор? 1
длина ключей RSA может быть от 1024 до 4096.
Какой размер ключа Вам необходим? (3072) 4096
Запрошенный размер ключа - 4096 бит
Выберите срок действия ключа.
         0 = не ограничен
      <n>  = срок действия ключа - n дней
      <n>w = срок действия ключа - n недель
      <n>m = срок действия ключа - n месяцев
      <n>y = срок действия ключа - n лет
Срок действия ключа? (0) 0
Срок действия ключа не ограничен
Все верно? (y/N) y

GnuPG должен составить идентификатор пользователя для идентификации ключа.

Ваше полное имя: lil000
Адрес электронной почты: arttiamashin22848@gmail.com
Примечание:
Вы выбрали следующий идентификатор пользователя:
    "lil000 <arttiamashin22848@gmail.com>"

Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход?
Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход? 0
Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход? 0
Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход? 0
Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход?
Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход? 0
Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход? (0)
Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход? o
```

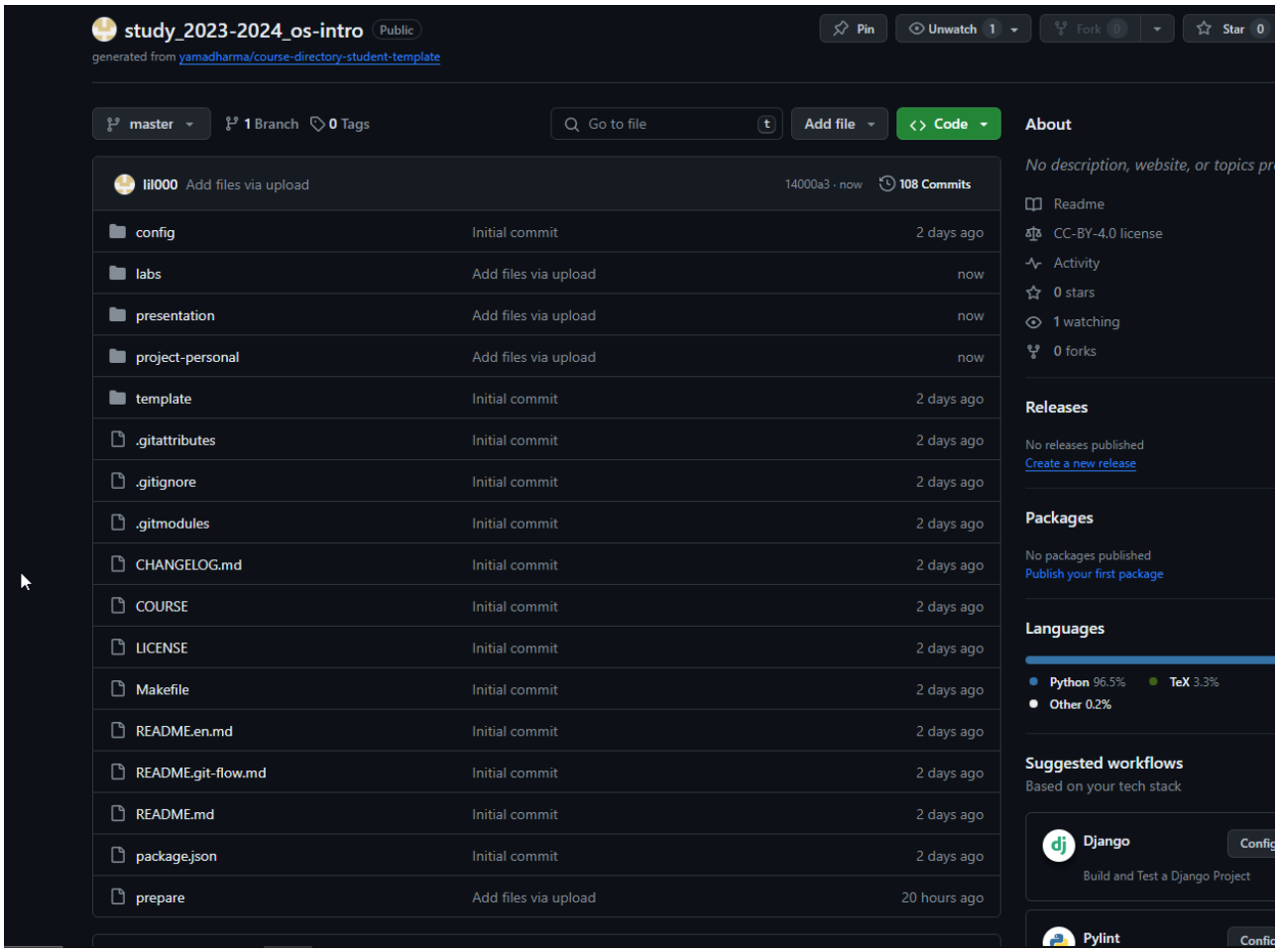
Скриншот: Выбираем нужные параметры ключа Добавляем ключ в гитхаб



6. Создаем репозитории курса по шаблону



Проверяем результат



Вывод:

Мы изучили идеаолгию и применение средств контроля версий.

Контрольные вопросы:

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

- . Системы контроля версий (Version Control Systems, VCS) - это программные инструменты, которые позволяют отслеживать изменения в коде или других файловых структурах, а также координировать работу нескольких разработчиков над одним проектом.

Основные задачи, для решения которых предназначены системы контроля версий:

- Организация коллективной разработки: VCS позволяют нескольким разработчикам работать над одним проектом одновременно, управляя конфликтами и комбинируя изменения.

- Отслеживание изменений: системы контроля версий позволяют отслеживать изменения в коде или других файлах, сохраняя лог изменений для последующего анализа.
- Восстановление предыдущих версий: VCS позволяют возвращаться к предыдущим версиям файлов в случае необходимости.
- Ветвление и слияние кода: системы контроля версий позволяют создавать отдельные ветки разработки, где разработчики могут вносить изменения без влияния на основной код, а затем сливать их обратно.
- Обеспечение целостности проекта: VCS обеспечивают сохранность данных и защиту от их случайного удаления или потери.

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

VCS (Version Control System) - система контроля версий, которая помогает отслеживать изменения в коде и управлять ими.

Хранилище (repository) - это место, где хранятся все файлы и история изменений проекта.

Commit - это операция, при которой изменения в рабочей копии добавляются в репозиторий.

История (history) - это список всех изменений, которые были сделаны в проекте, включая информацию о том, кто и когда внес изменения.

Рабочая копия (working copy) - это копия проекта сделанных изменений, которая находится на компьютере пользователя.

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

```
Централизованные VCS имеют одно центральное хранилище, к которому подключаются все пользователи.
```

Пример централизованной системы

```
- **SVN (Subversion).**
```

Децентрализованные VCS позволяют работать с несколькими копиями репозитория, каждая из которых может быть независимой.

Примеры децентрализованных систем:

```
- **Git, Mercurial.**
```

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

```
При единоличной работе с хранилищем в VCS пользователь может создать копию
репозитория на своем компьютере, вносить изменения в код, коммитить их и при
необходимости возвращаться к предыдущим версиям.
```

5. Опишите порядок работы с общим хранилищем VCS.

```
Для работы с общим хранилищем VCS необходимо сначала склонировать репозиторий на
свой компьютер, вносить изменения, коммитить их и отправлять на удаленный
репозиторий при необходимости.
```

6. Каковы основные задачи, решаемые инструментальным средством git?

```
Основные задачи инструмента git: отслеживание изменений в коде, управление ветками
и слияниями, работа с удаленными репозиториями, возврат к предыдущим версиям.
```

7. Назовите и дайте краткую характеристику командам git.

git add - добавить файлы в индекс

git commit - создать коммит

git push - отправить изменения на удаленный репозиторий

git pull - получить изменения с удаленного репозитория

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

```
При работе с локальными репозиториями можно создавать новые ветки, коммитить
изменения и проводить операции слияния. При работе с удаленными репозиториями
можно отправлять изменения на сервер, получать изменения с сервера, и работать с
ветками.
```

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

Ветви **(branches)** в **git** позволяют работать с разными версиями кода параллельно. Они помогают изолировать различные фичи и эксперименты, и в случае необходимости вносить изменения в одной ветке, не затрагивая другие.

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

Git позволяет игнорировать некоторые файлы при коммите с помощью файла **.gitignore**, в котором можно указать шаблоны файлов или каталогов, которые необходимо игнорировать при добавлении в репозиторий.