РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

дисциплина: Операционные системы

Студент: Желдакова Виктория Алексеевна

Студ. Билет №1032216445

Группа: НФИбд-01-21

МОСКВА

2022 г.

Управление версиями

Цель работы:

1) Изучить идеологию и применение средств контроля версий.
2) 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.

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. 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, а производить так называемую дельтакомпрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.

Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить (слить) изменения, сделанные разными участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависимости от настроек блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, привилегированный доступ только одному пользователю, работающему с файлом.

Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например, они могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Кроме того, обычно доступна информация

о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить.

В отличие от классических, в распределённых 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.

Выполнение работы:

Во-первых, мы создали учётную запись на github.com и заполнили основные данные о себе.

Затем установили git-flow (Рис.1.1, Рис.1.2) и gh (Рис.2).

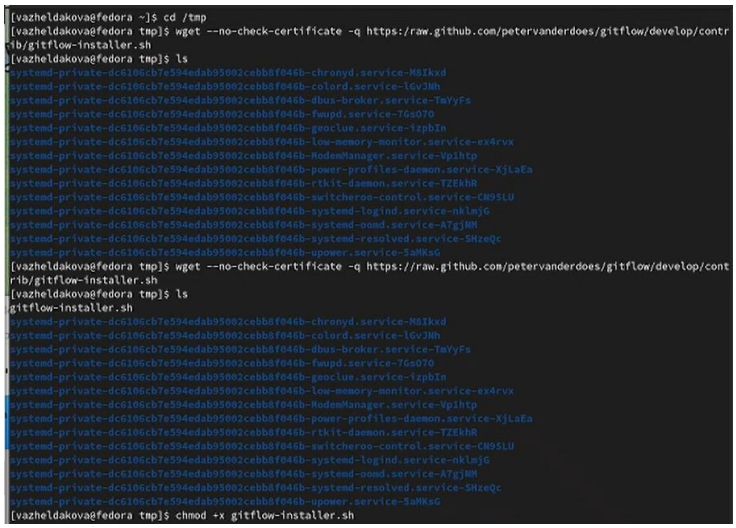

Рис.1.1 Скачивание установщика git-flow и изменение его прав доступа

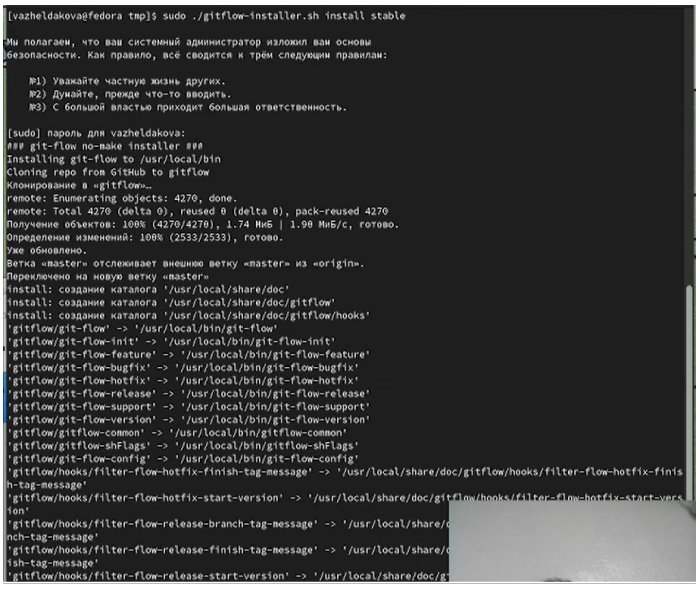

Рис.1.2 Установка git-flow

Рис.2 Установка gh

Затем задали имя и email владельца репозитория, настроили utf-8 в выводе сообщений git, а также верификацию и подписание коммитов git. Дали имя «master» начальной ветке. Настроили параметры autocrlf и safecrlf.

```
[vazheldakova@fedora tmp]$ git config --global user.name Viktoria Zheldakova
[vazheldakova@fedora tmp]$ git config --global user.email v.zhel3@ynadex.ru
[vazheldakova@fedora tmp]$ git config --global core.quotepath false
[vazheldakova@fedora tmp]$ git config --global init.defaultBranch master
[vazheldakova@fedora tmp]$ git config --global core.autocrif input
[vazheldakova@fedora tmp]$ git config --global core.safecrlf warn
[vazheldakova@fedora tmp]$ git config --global core.autocrlf input
```

Рис.3 Базовая настройка git

Создали ключи SSH по алгоритмам RSA и ed25519.

```
[vazheldakova@fedora tmp]$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vazheldakova/.ssh/id_rsa):
Created directory '/home/vazheldakova/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vazheldakova/.ssh/id_rsa
Your public key has been saved in /home/vazheldakova/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:SUPm0kye5JAXz1MeSoj9m2Y7gbH11hN9V52cYrnbzjg vazheldakova@fedora
The key's randomart image is:
+---[RSA 4096]----+
|      .+*o. o o +|
|      o%+= + = +o|
|      ..X.= o o..|
|       o.oo. .. +|
|        S= + .o.o|
|        o * o.o. |
|         o +  +. |
|          o  E o |
|             .  .|
+----[SHA256]-----+
```

Рис.4.1 Создание SSH ключа по алгоритму RSA

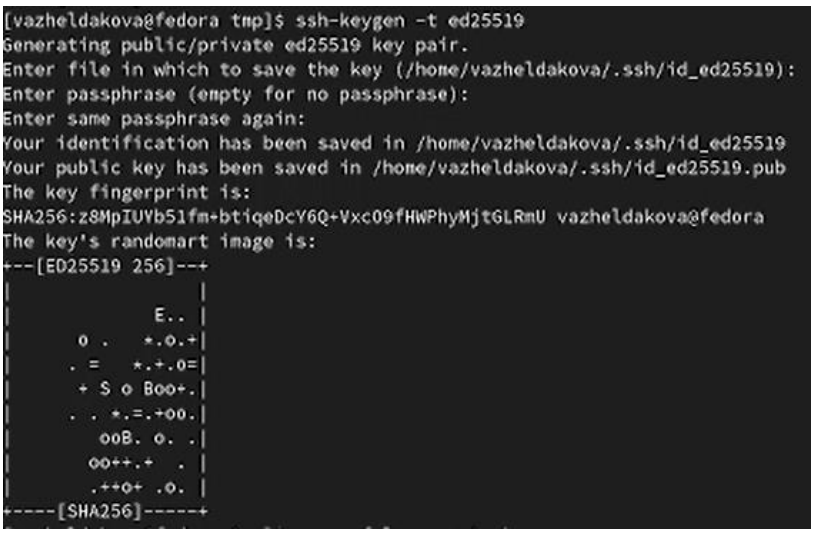

Рис.4.2 Создание SSH ключа по алгоритму ed25519

Сгенерировали GPG ключ.

Рис.5.1 Настройка генерации GPG ключа

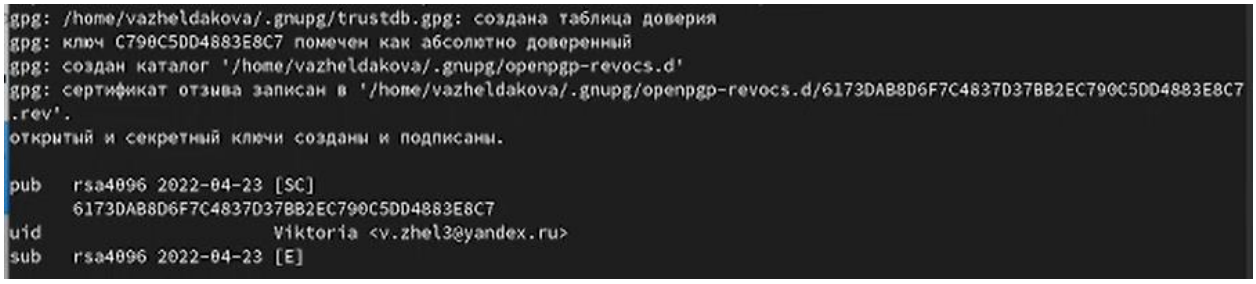

Рис.5.2. Результат генерации GPG ключа

Вывели список ключей и скопировали отпечаток приватного ключа, затем скопировали сгенерированный ключ в буфер обмена и вставили полученный ключ в поле ввода в настройках GitHub. (Так же пришлось установить утилиту xclip для копирования в буфер).

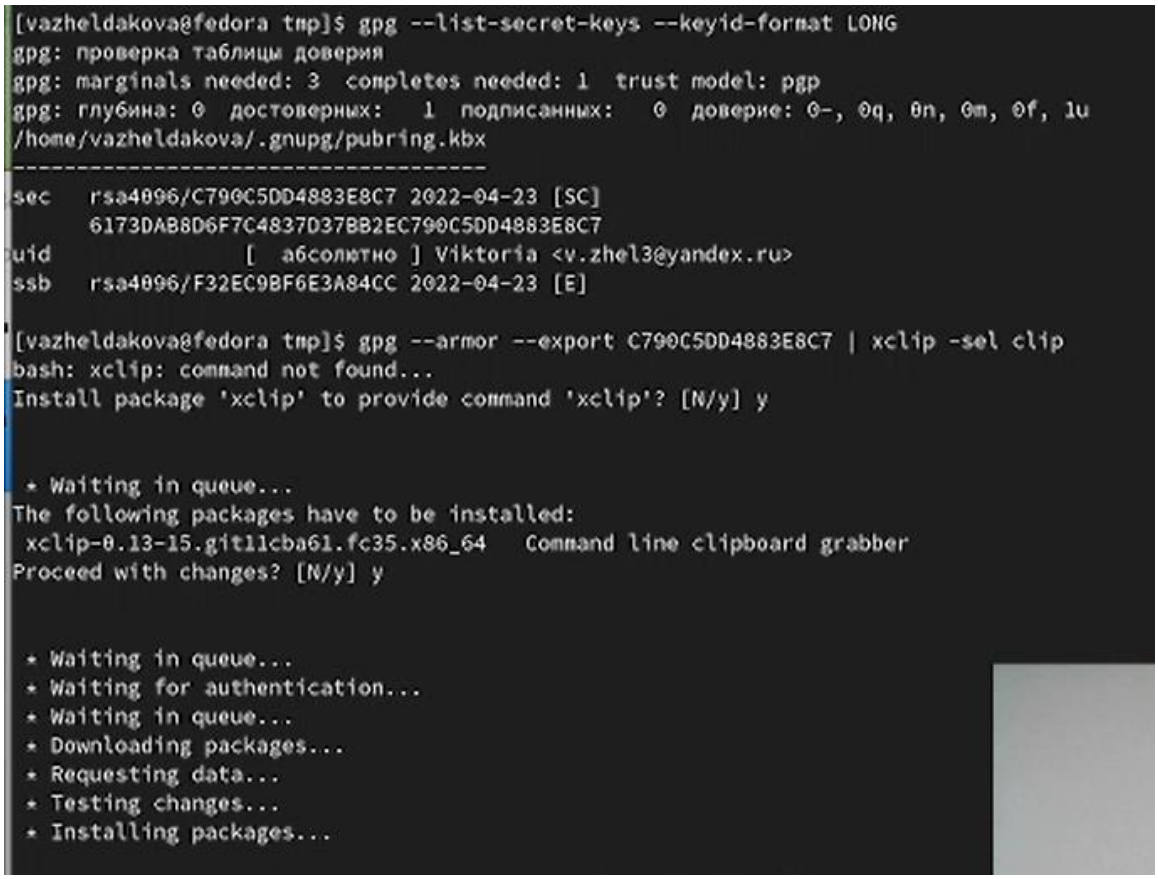

Рис.6.1 Вывод списка ключей и установка xclip

Рис.6.2 Добавление ключа в буфер обмена

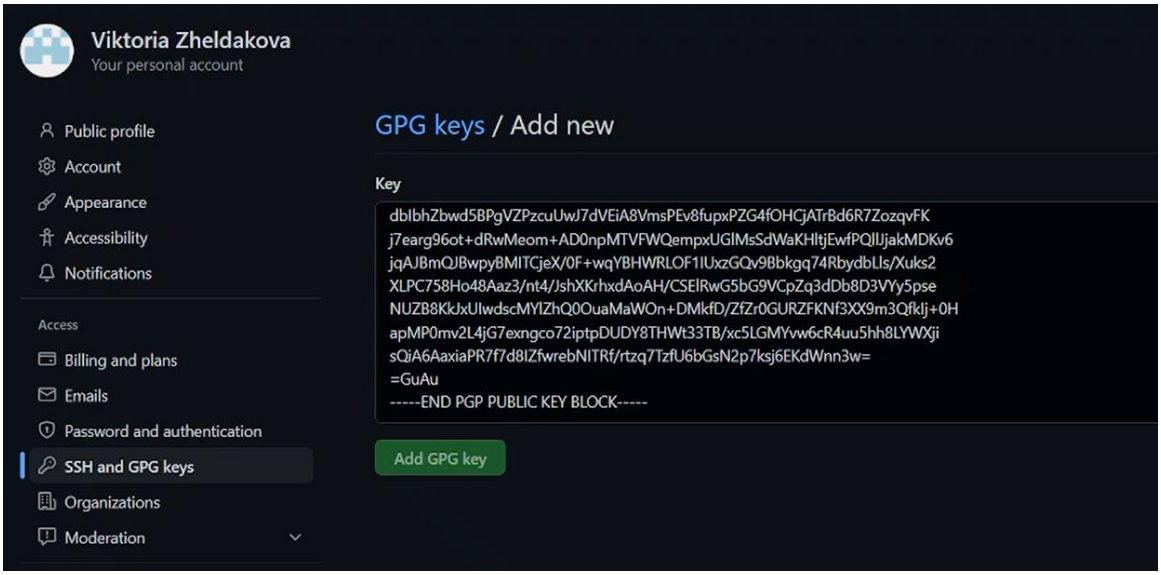

Рис.6.3 Добавление ключа в настройки GitHub

Настроили автоматическую подпись коммитов git.

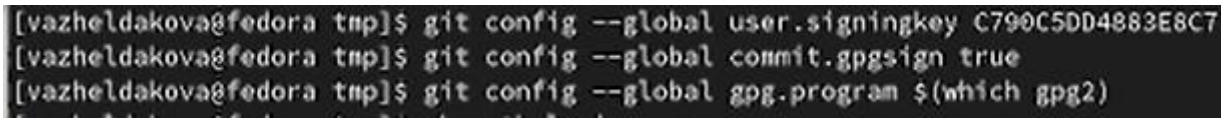

Рис. 7 Настройка подписи коммитов

Затем авторизовались на github.com

Рис.8 Авторизация

Создали каталог «Операционные системы», в нём создали репозиторий и клонировали шаблон.

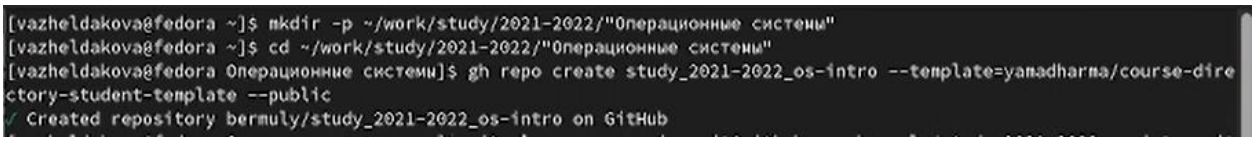

Рис.9.1 Создание каталога и репозитория в нём

Рис.9.2 Клонирование шаблона

Перешли в каталог курса, удалили лишние файлы, создали необходимые каталоги и отправили файлы на сервер.

```
[vazheldakova@fedora Операционные системы]$ cd ~/work/study/2021-2022/"Операц
[vazheldakova@fedora os-intro]$ rm package.json
[vazheldakova@fedora os-intro]$ make COURSE=os-intro
[vazheldakova@fedora os-intro]$ git add .
[vazheldakova@fedora os-intro]$ git commit -am 'feat(main): make course struc
```

Рис.10.1 Настройка каталога курса

Рис.10.2 Отправка изменений в репозиторий

Вывод:

Изучили идеологию и применение средств контроля версий. Получили практические навыки по работе с git.

Контрольные вопросы:

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

Система контроля версий (Version Control System, VCS) представляет собой программное обеспечение, которое позволяет отслеживать изменения в документах, при необходимости производить их откат, определять, кто и когда внес исправления и т.п.

Она предназначается для задач, когда необходимо часто вносить изменения в проект, но при этом должна сохраняться работоспособность. VCS отслеживает изменения в файлах, предоставляет возможности для создания новых и слияние существующих ветвей проекта, производит контроль доступа пользователей к проекту, позволяет откатывать исправления и определять кто, когда и какие изменения вносил в проект.

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

Основным понятием VCS является репозиторий (repository) – специальное хранилище файлов и папок проекта, изменения в которых отслеживаются. В распоряжении разработчика имеется так называемая “рабочая копия” (working copy) проекта, с которой он непосредственно работает. Рабочую копию необходимо периодически синхронизировать с репозиторием, эта операция предполагает отправку в него изменений, которые пользователь внес в свою рабочую копию (такая операция называется commit), при запуске команды git log можно получить список последних коммитов в обратном хронологическом порядке (историю).

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

Централизованные системы контроля версий представляют собой приложения типа клиент-сервер, когда репозиторий проекта существует в единственном экземпляре и хранится на сервере. Доступ к нему осуществлялся через специальное клиентское приложение. В качестве примеров таких программных продуктов можно привести CVS, Subversion.

Распределенные системы контроля версий (Distributed Version Control System, DVCS) позволяют хранить репозиторий (его копию) у каждого разработчика, работающего с данной системой. При этом можно выделить центральный репозиторий (условно), в который будут отправляться изменения из локальных и, с ним же эти локальные репозитории будут синхронизироваться. Две наиболее известные DVCS – это Git и Mercurial.

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

Разработчик работает с начальной веткой, для разных частей проекта может создать отдельные ветки. После внесения изменений разработчик

коммитит (commit) и пушит (push) их, то есть сохраняет изменения в общем хранилище.

5. Опишите порядок работы с общим хранилищем VCS.

Каждый разработчик работает над отдельной частью проекта в своей ветке или над изменениями, сделанными другими разработчиками. После внесения изменений разработчик коммитит (commit) и пушит (push) их в общее хранилище. В конце работы необходимо выполнить слияние (merge) веток, например, с начальной.

6. Каковы основные задачи, решаемые инструментальным средством git?

С помощью Git-а вы можете откатить свой проект до более старой версии, сравнивать, анализировать или сливать свои изменения в репозиторий.

7. Назовите и дайте краткую характеристику командам 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 изменения:

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

git commit

– создание новой ветки, базирующейся на текущей:

git checkout -b имя_ветки

– переключение на некоторую ветку:

git checkout имя_ветки

– 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

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

Первый разработчик начал верстать проект и сделал первые коммиты, в которых зафиксировал изменения в определённом файле. После того как он сделал два коммита, он захотел отправить свои изменения в удалённый репозиторий. Чтобы их передать, разработчик выполнил команду git push.

После чего в облаке появилось две версии проекта. То есть он отправил не только финальную версию проекта, но и все сохранённые изменения.

После пуша данные синхронизировались с удалённым репозиторием. Для того, чтобы получить эти изменения, другой разработчик выполняет команду git pull и получает все изменения из облака к себе на компьютер. Таким образом, состояние проекта у обоих синхронизировалось, и они могут дальше продолжить работать над ним.

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

Branches (ветки) представляют собой отдельные копии проекта. Они используются для одновременной разработки отдельных частей проекта.

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

Для игнорирования файлов при commit используют файл .gitignor. В него записываются файлы, которые не нужны для проекта, например, временные файлы, создаваемые редакторами, системные файлы и скомпилированный код.