**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**Отчет по лабораторной работе №2**

*Дисциплина: Операционные системы*

Студент: Парфенова Елизавета Евгеньевна

Группа: НФИбд-02-21

**Москва**

2022 г.

**Управление версиями**

**Цель работы:**

– Изучить идеологию и применение средств контроля версий.

– Освоить умения по работе с git.

**Ход работы:**

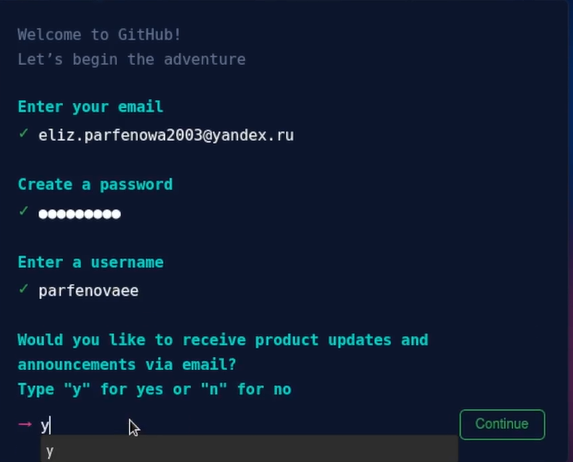
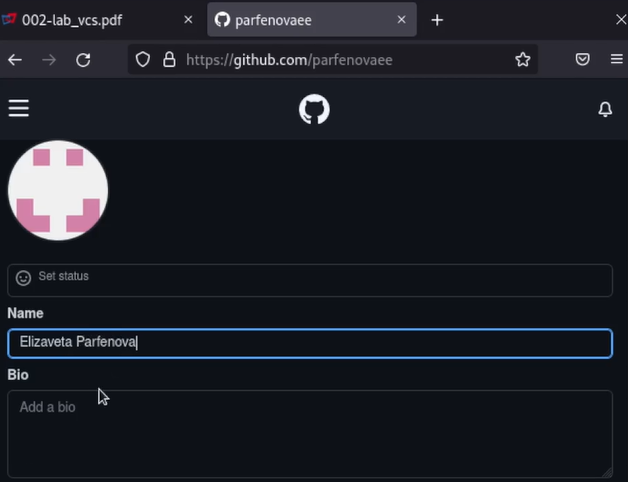
Вначале я перешла по ссылке <https://github.com> и создала учетную запись на github. (Рис. 1) После подтверждения кодом, который пришел на электронную почту, я настроила основные данные аккаунта. (Рис.2)

Рис. 1 Регистрация на github Рис. 2 Основные данные аккаунта

После я перешла в терминал и установку программного обеспечения.

Вначале я установила git-flow в Fedora Linux. Для этого я ввела три команды, так как ПО необходимо проводить вручную. (Рис. 3)

Команды:

* ***wget --no-check-certificate -q*** [***https://raw.github.com/petervanderdoes /gitflow/develop/contrib/gitflow-installer.sh***](https://raw.github.com/petervanderdoes%20/gitflow/develop/contrib/gitflow-installer.sh)
* ***chmod +x gitflow-installer.sh***
* ***sudo ./gitflow-installer.sh install stable***

Далее я провела установку gh в Fedora Linux с помощью команды ***sudo dnf install gh.*** (Рис. 4)

Следующим шагом я совершила базовую настройку git.

Изображение выглядит как текст

Автоматически созданное описаниеРис. 3 Установка git-flow

Изображение выглядит как текст, счетчик, устройство, панель управления

Автоматически созданное описаниеРис. 4 Установка gh

С помощью двух команд я задала имя и email владельца репозитория:

* ***git config --global user.name ""***
* ***git config --global user.email "eliz.parfenowa2003@yandex.ru"***

Далее я настроила utf-8 в выводе сообщений git, используя команду ***git config --global core.quotepath false.*** После этого командой ***git config --global init.defaultBranch master*** я задала имя начальной ветки (master). Осталась настройка последних двух параметров. Первый параметр autocrlf я настроила командой ***git config --global core.autocrlf input***, а второй параметр safecrlf – командой ***git config --global core.safecrlf warn.*** (Рис. 5)

Изображение выглядит как текст, табло, табличка

Автоматически созданное описаниеРис. 5 Базовые настройки git

Следующий шаг – создание ключа ssh. Вначале командой ***ssh-keygen -t rsa -b 4096*** я создаю ключ по алгоритму rsa размером 4096 бит (Рис.6.1), а после командой ***ssh-keygen -t ed25519*** – по алгоритму ed25519. (Рис. 6.2)

Изображение выглядит как текст

Автоматически созданное описание

Рис.6.1 Создание ключа ssh по алгоритму rsa

Изображение выглядит как текст

Автоматически созданное описание

Рис. 6.2 Создание ключа ssh по алгоритму ed25519

Следующим шагом я создала ключ pgp. Вначале необходимо было сгенерировать ключ с помощью команды ***gpg --full-generate-key.*** Далее терминал предложил выбрать мне некоторые опции, и я выбрала их в соответствии с требованиями лабораторной работы. (тип RSA and RSA; размер 4096; не истекающий срок действия). Также я заполнила необходимую личную информацию (имя и адрес электронной почты). (Рис 7)

Изображение выглядит как текст

Автоматически созданное описаниеРис. 7 Создание ключа pgp

Далее созданный ключ я добавила в Github. Для того чтобы это сделать вначале было необходимо создать отпечаток приватного ключа. С помощью команды ***gpg --list-secret-keys --keyid-format LONG*** я вывела список ключей и нашла нужный (Рис.8). Скопировала его и вставила в данную комнаду ***gpg --armor –export <PGP Fingerprint> | xclip -sel clip*** вместо <PGP Fingerprint>. (Рис.9) Это нужно было для того, чтобы скопировать ключ в буфер обмена. После я зашла на Github и добавила ключ туда, во вкладку GPG Keys. (Рис. 10)

Далее я настроила автоматические подписи коммитов git. Для этого используем введенный email. (Рис 11)

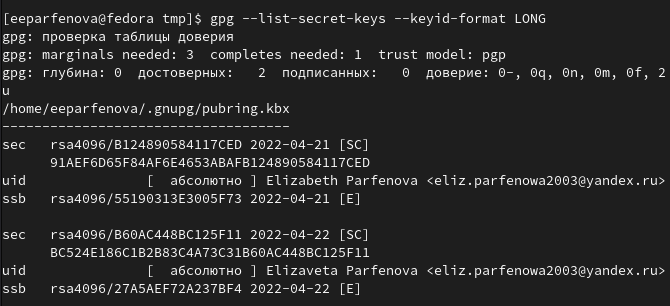
******Рис. 8 Вывод списка ключей

Рис. 9 Копирование нужного ключа в буфер обмена

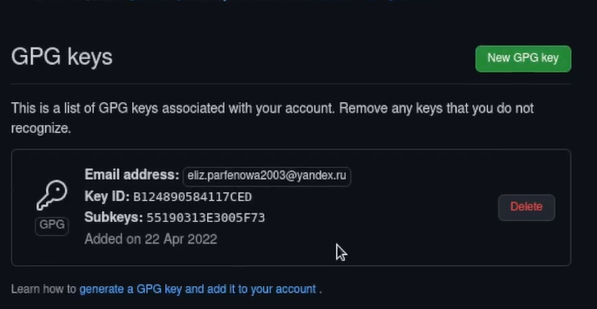
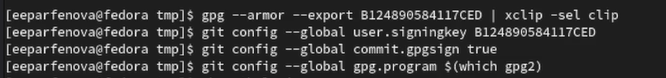
Рис. 10 Добавление ключа на Github

Рис. 11 Настройка автоматических подписей коммитов git

После необходимо было настроить gh. Вначале с помощью команды ***gh auth login*** я авторизовалась. После ответила на вопросы, заданные в терминале. Он вывел код и запустила страницу браузера, где это необходимо было ввести. После ввода все завершилось успешно. (Рис. 12)

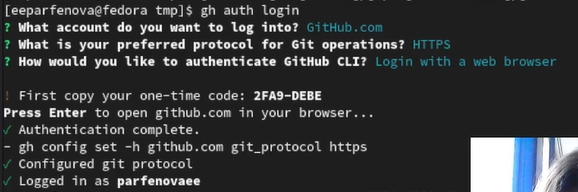


Рис. 12 Настройка gh

Следующим шагом нужно было создать репозиторий курса на основе представленного в лабораторной работе шаблона. Создание репозитория представляло собой серию вот таких команд:

* ***mkdir -p ~/work/study/2021-2022/"Операционные системы"***
* ***cd ~/work/study/2021-2022/"Операционные системы"***
* ***gh repo create study\_2021-2022\_os-intro***
* ***git clone –recursive*** [***git@github.com***](mailto:git@github.com)***: parfenovaee/study\_2021-2022\_os-intro.git os-intro***

Однако, чтобы последняя команда прошла успешно, нужно выполнить дополнительные действия. Я загрузила ssh ключ на Github. Для этого я использовала команду ***cat ~/.ssh/id\_rsa.pub | xclip -sel clip,*** которая скопировала существующий ключ в буфер обмена. (Рис. 13) После переходим на Github в уже знакомую у , где добавляли GPG ключ, и добавляем SSH ключ в нужное окно. (Рис 14)

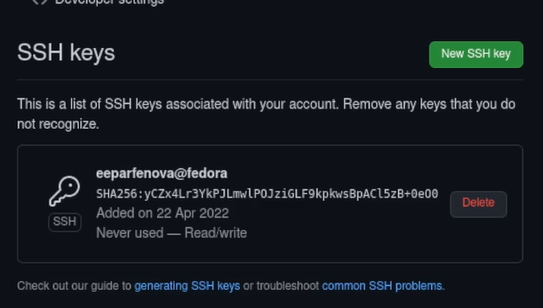
Рис. 13 Копирование ключа в буфер обмена

Рис. 14 Добавление ключа на Github

Теперь все команды по созданию репозитория курса могут быть выполнены успешно. (Рис. 15)

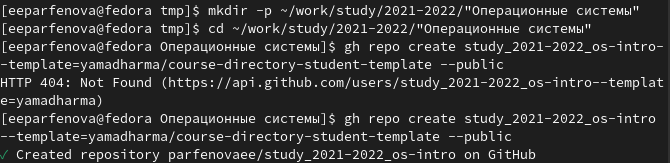
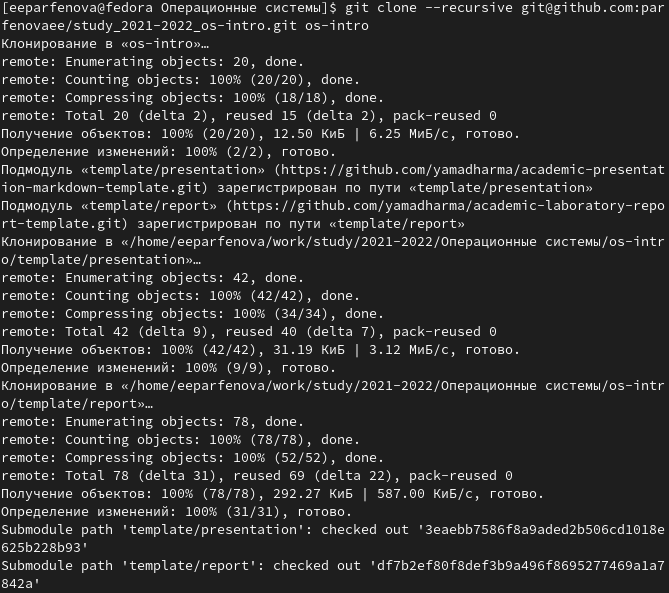


Рис. 15 Создание репозитория курса на основе шаблона

Последним шагом является настройка каталога курса. Начала я ее с команды ***cd ~/work/study/2021-2022/"Операционные системы"/os-intro***, которая позволила перейти в каталог. После я удалила лишние файлы командой ***rm package.json***. Создала необходимые каталоги, используя команду ***make COURSE=os-intro***. И отправила файлы на сервер следующими командами (Рис 16.1):

* ***git add .***
* ***git commit -am 'feat(main): make course structure'*** (эта команда требует кодовой фразы, которую мы вводили ранее)
* ***git push***

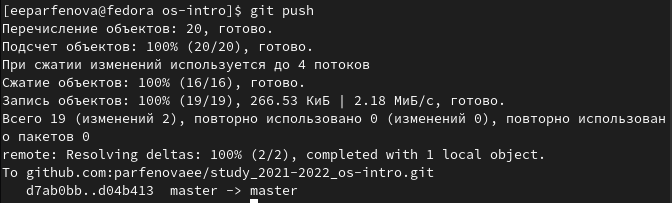
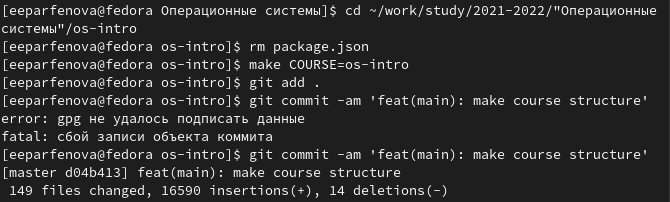
******

Рис. 16. Настройка каталога курса

**Вывод:** Мы изучили идеологию и применение средств контроля версий, а также освоили умения по работе с git.

**Контрольные вопросы:**

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

Система контроля версий (VCS) - [программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения [исходных кодов](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов.

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

Хранилище – репозиторий  - место хранения всех версий и служебной информации.

Commit - **это** команда для записи индексированных изменений в репозиторий.

История – место, где сохраняются все коммиты, по которым можно посмотреть данные о коммитах.

Рабочая копия – текущее состояние файлов проекта, основанное на версии, загруженной из хранилища.

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

Централизованные системы – это системы, в которых одно основное хранилище всего проекта, и каждый пользователь копирует необходимые ему файлы, изменяет и вставляет обратно. Пример – Subversion.

Децентрализованные системы – система, в которой каждый пользователь имеет свой вариант репозитория и есть возможность добавлять и забирать изменения из репозиториев. Пример – Git.

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

В рабочей копии, которую исправляет человек, появляются правки, которые отправляются в хранилище на каждом из этапов. То есть в правки в рабочей копии появляются, только если человек делает их (отправляет их на сервер) и никак по-другому .

1. Опишите порядок работы с общим хранилищем VCS.

Если хранилище общее, то в рабочую копию каждого, кто работает над проектом, приходят изменения, отправленные на сервер одним из команды. Рабочая правка каждого может изменяться вне зависимости от того, делает ли конкретный человек правки или нет.

1. Каковы основные задачи, решаемые инструментальным средством git?

У Git две основных задачи: первая — хранить информацию обо всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом.

1. 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

– удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm имена\_файлов

– 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: 1 git push origin имя\_ветки

– слияние ветки с текущим деревом: 1 git merge --no-ff имя\_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

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

Работа с удаленным репозиторием: git remote – просмотр списка настроенных удаленных репозиториев.

Работа с локальным репозиторием: git status - выводит информацию обо всех изменениях, внесенных в дерево директорий проекта по сравнению с последним коммитом рабочей ветки

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

Ветка (англ. branch) — это последовательность коммитов, в которой ведётся параллельная разработка какого-либо функционала. Ветки нужны, чтобы несколько программистов могли вести работу над одним и тем же проектом или даже файлом одновременно, при этом не мешая друг другу. Кроме того, ветки используются для тестирования экспериментальных функций: чтобы не повредить основному проекту, создается новая ветка специально для экспериментов.

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

Игнорируемые файлы — это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты.  В Git нет специальной команды для указания игнорируемых файлов: вместо этого необходимо вручную отредактировать файл . Временно игнорировать изменения в файле можно командой ***git update-index-assumeunchanged<file>***