Операционные системы

Отчет по лабораторной работе №2

Матюшкин Денис Владимирович(НПИбд-02-21)

Содержание

# 1 Цель работы:

* Изучить идеологию и применение средств контроля версий.
* Освоить умения по работе с git.

# 2 Ход работы:

1. Создадим учетную версию на https://github.com и заполним основные данные (рис. 1). Примечание: я ранее регистрировался на данном сайте, поэтому не буду повторяться.

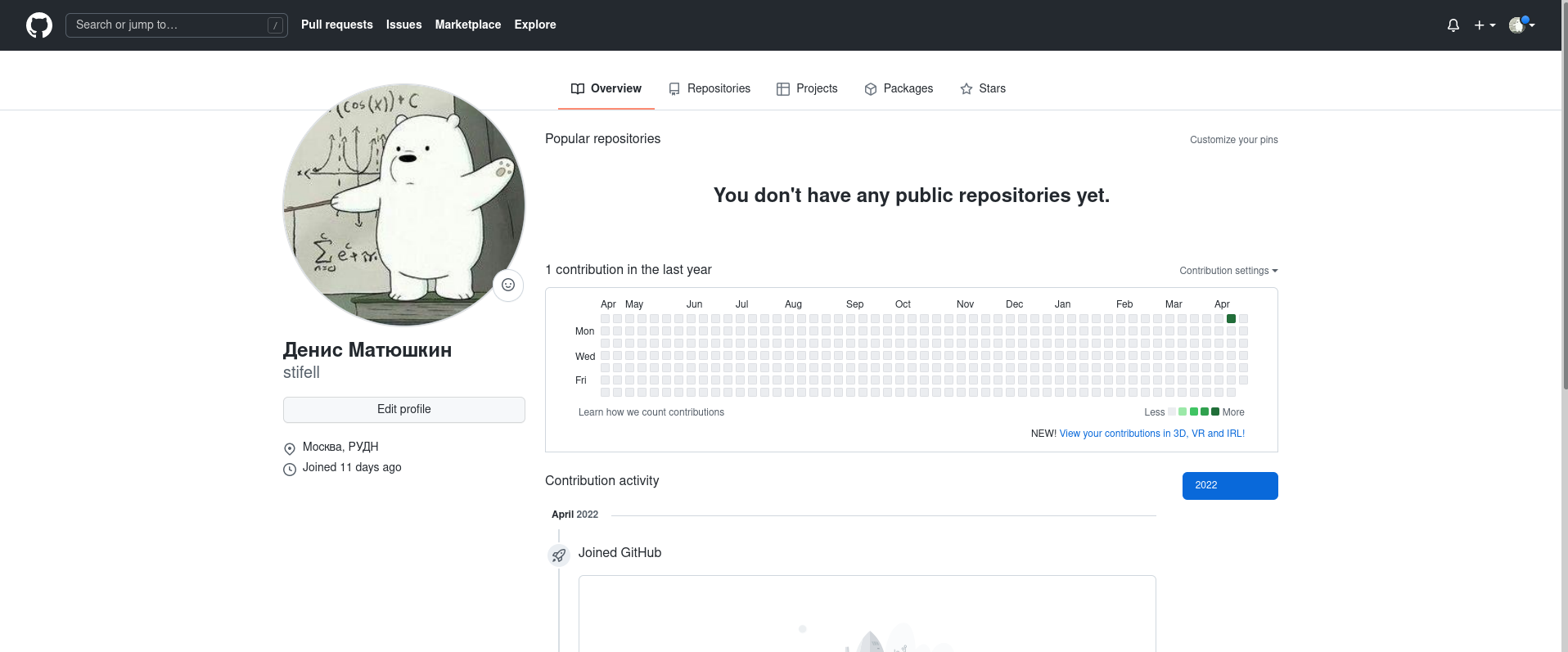


Рис. 1: Создание учетной записи на Github

1. Установим программное обеспечение git-flow через терминал (рис. 2).

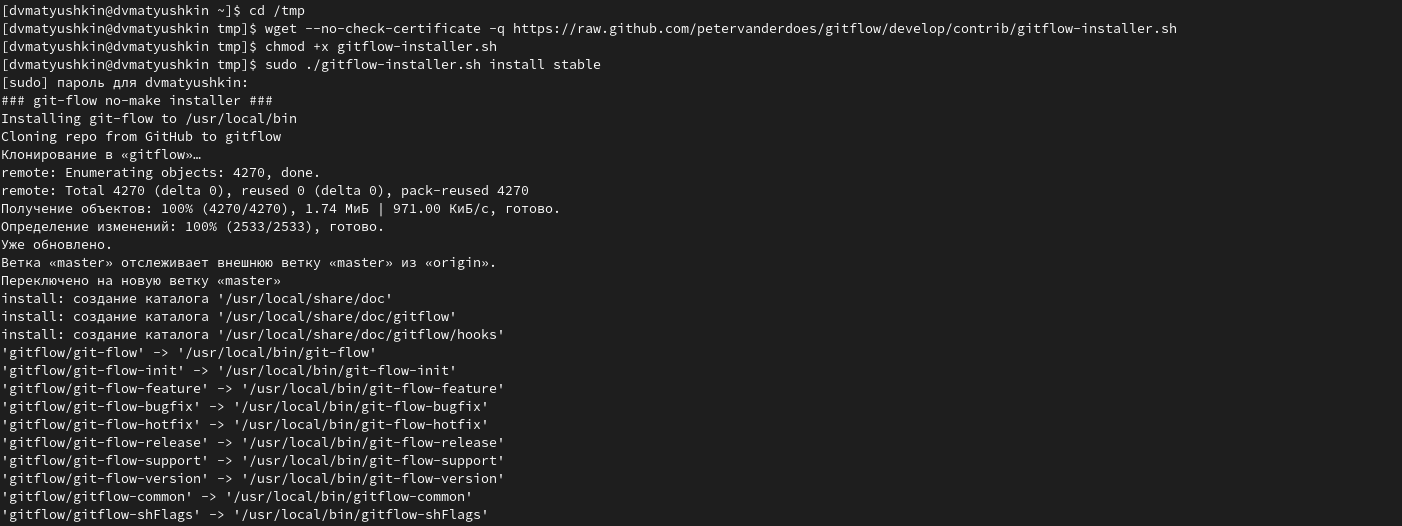


Рис. 2: Установка git-flow на Linux Fedora

1. Установим программное обеспечение gh через терминал (рис. 3).

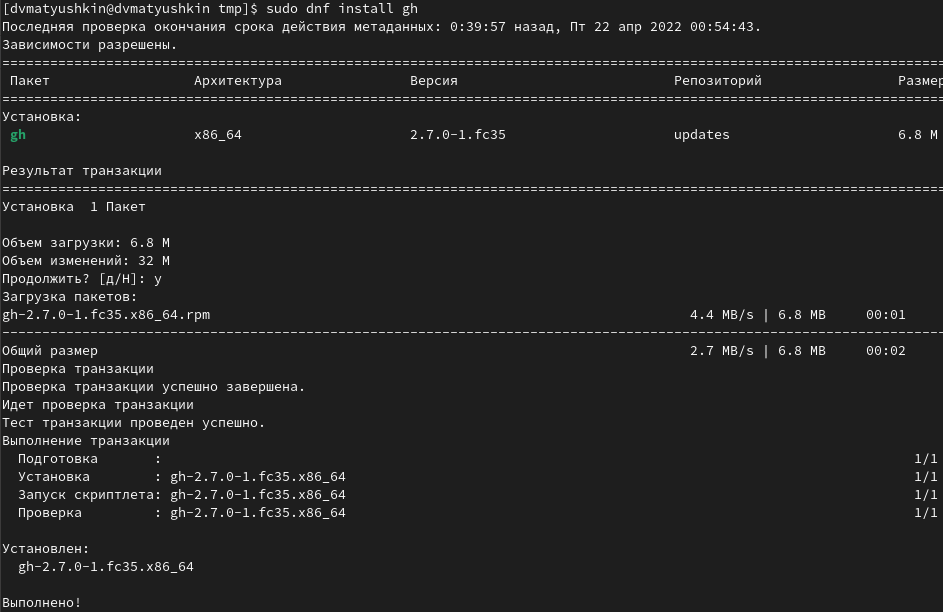


Рис. 3: Установка gh на Linux Fedora

1. Совершим базовую настройку git (рис. 4):
   1. Зададим имя и email владельца репозитория.
   2. Настроим utf-8 в выводе сообщений git.
   3. Настроим верификацию и подписание коммитов git.
   4. Зададим имя начальной ветки (будем называть её master).
   5. Зададим параметр autocrlf и safecrlf.

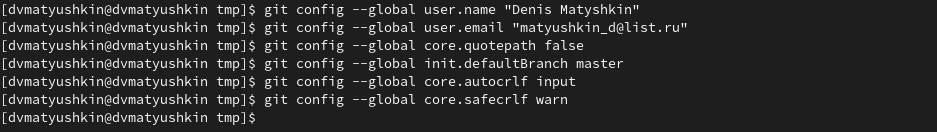


Рис. 4: Базовая настройка git

1. Создадим ключ SSH (рис. 5):
   1. по алгоритму rsa с ключём размером 4096 бит;
   2. по алгоритму ed25519.

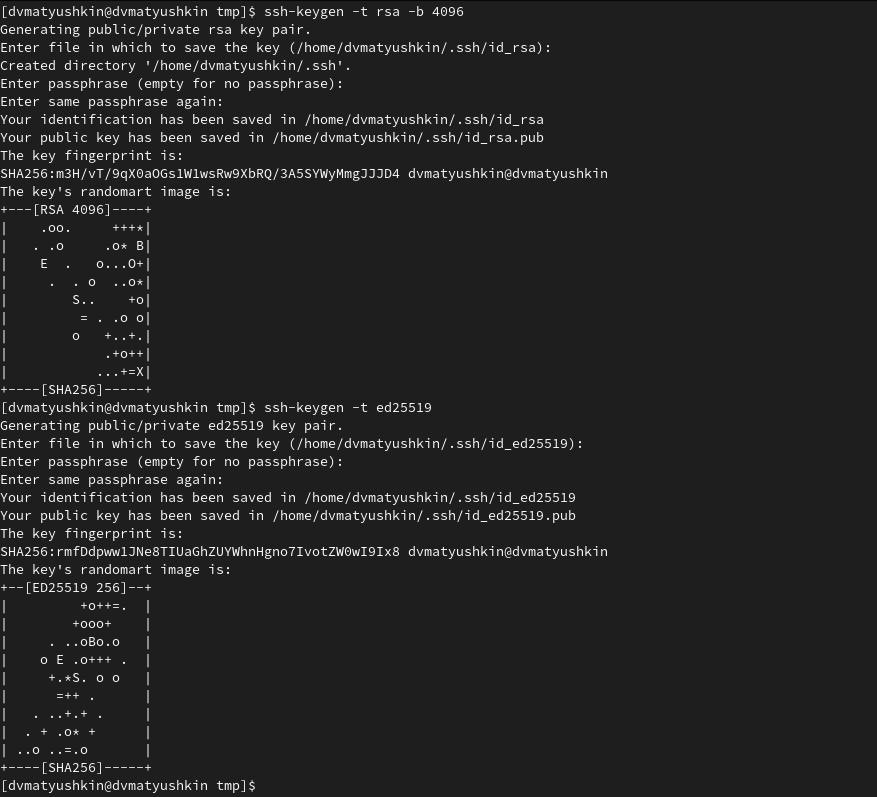


Рис. 5: Создание ключа ssh

1. Создадим ключ GPG (рис. 6.1 и 6.2). Выберем опции, описанные в лабораторной: – тип RSA and RSA;
   * размер 4096; – выберите срок действия; значение по умолчанию — 0 – GPG запросит личную информацию, которая сохранится в ключе. – Имя. – Адрес электронной почты.

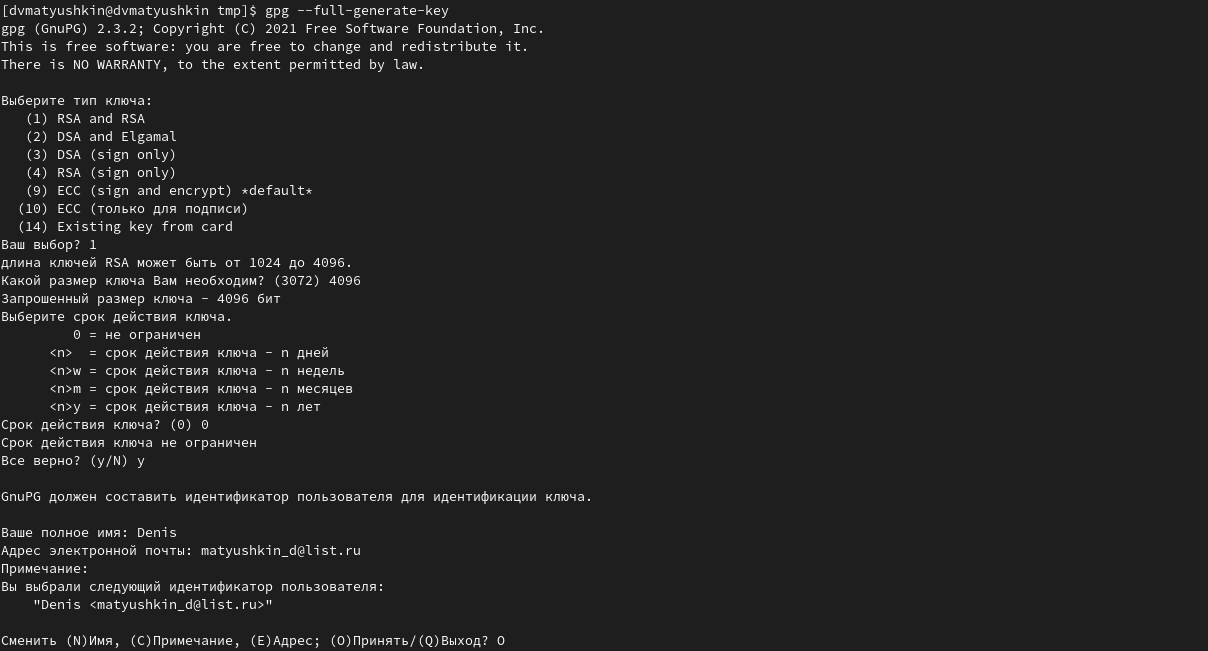


Рис. 6.1: Создание ключа gpg



Рис. 6.2: Продолжение вывода

1. Выводим список ключей и копируем отпечаток приватного ключа (рис. 7). Отпечаток ключа – A7BCA013E5F9D27B

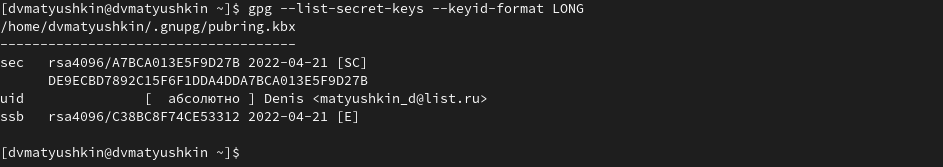


Рис. 7: Вывод списка приватных ключей

1. Скопируем наш сгенерированный GPG ключ в буфер обмена (рис. 8.1) и вставим его в настройках личнего кабинета Github (рис. 8.2).

Рис. 8.1: Копирования gpg ключа в буфер обмена

Рис. 8.1: Копирования gpg ключа в буфер обмена

1. Сгенерированный SSH ключ и скопируем его в буфер обмена (рис. 9) и вставим его в настройках личнего кабинета Github (рис. 8.2).

Рис. 9: Копирования ssh ключа в буфер обмена

Рис. 9: Копирования ssh ключа в буфер обмена

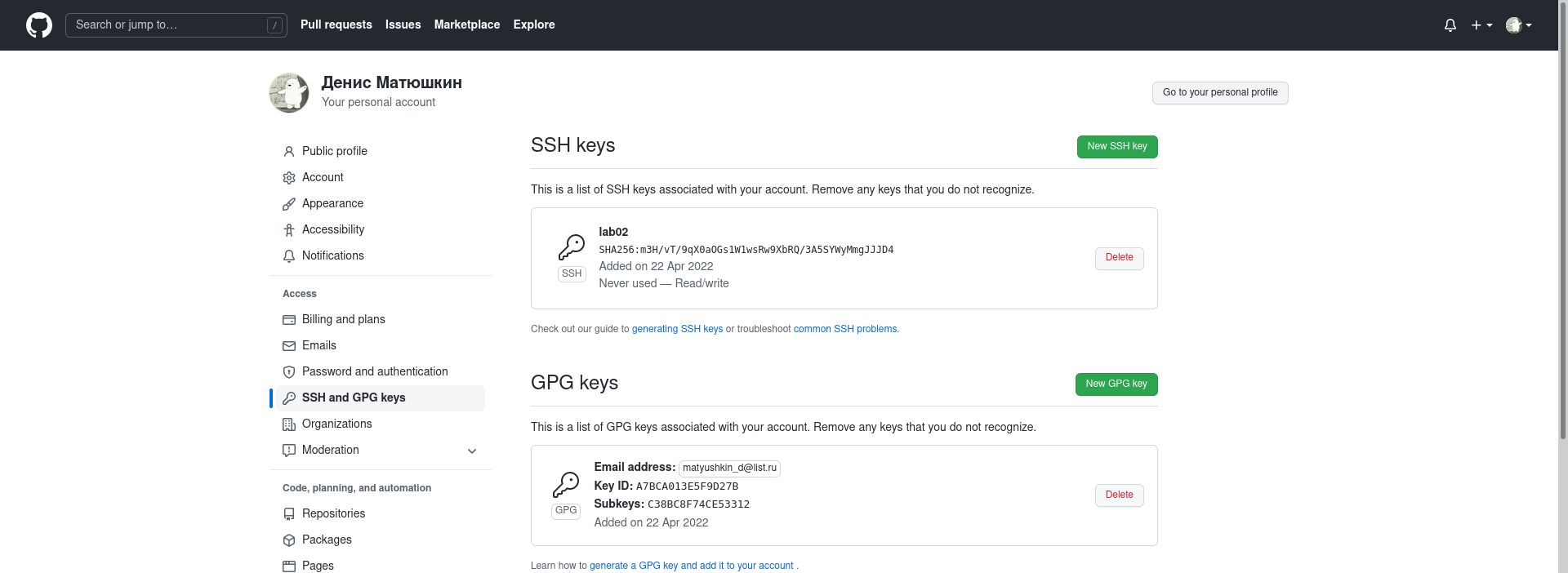


Рис. 8.2: Заполнение ключей SSH и GPG в личном аккаунте

1. Настроим автоматические подписи коммитов Git. Используя введенный email, укажем Git применять его при подписи коммитов (рис. 10).

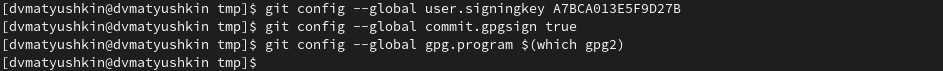


Рис. 10: Настройка автоматических подписей коммитов Git

1. Настроим gh. Для начала необходимо авторизоваться, можно через браузер (рис. 11).

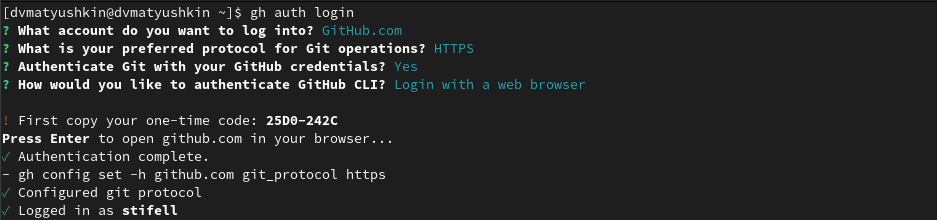


Рис. 11: Настройка gh

1. Создадим репозитория курса на основе шаблона. Для этого создадим каталоги work/study/2021-2022/”Операционные системы” (рис. 12)

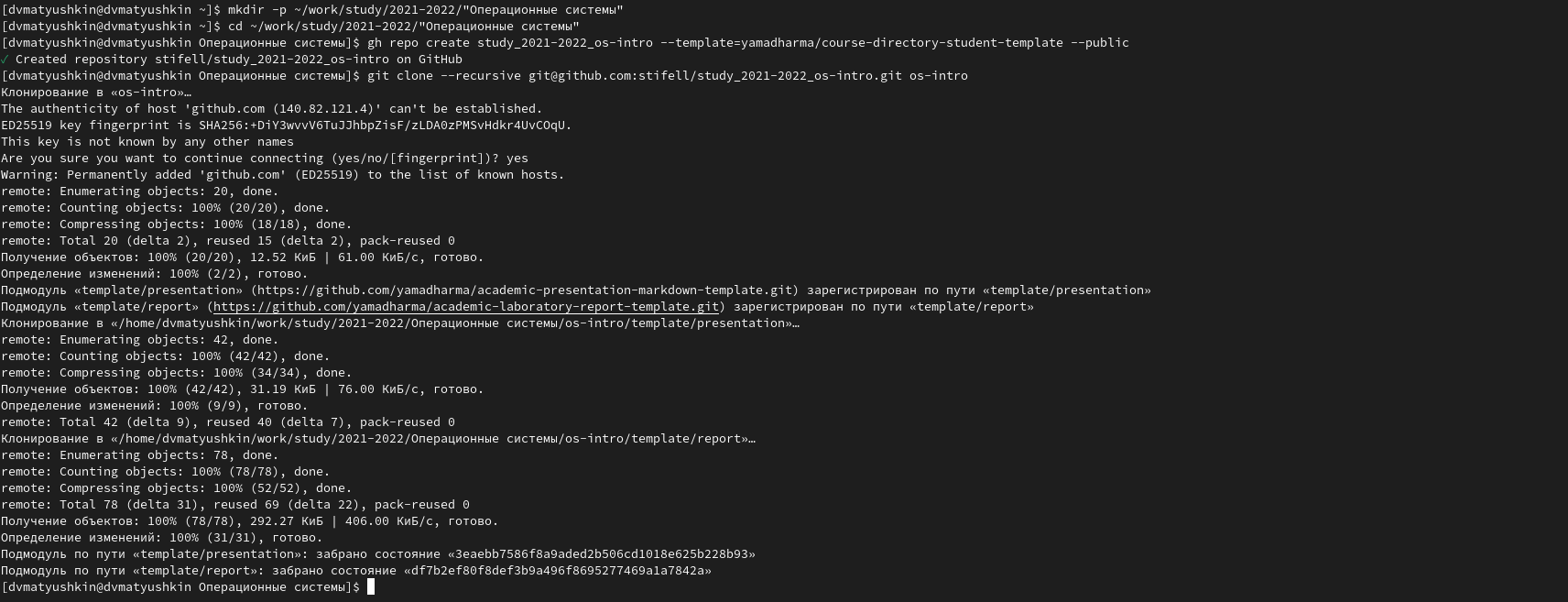


Рис. 12: Создание репозитория курса

1. Настроим каталог курса (рис. 13.1, 13.2 и 13.3). Для этого:
   1. Перейдем в каталог курса.
   2. Удалим лишние файлы.
   3. Создадим необходимые каталоги.
   4. Отправим файлы на сервер.

Рис. 13.1: Переход в каталог, удаление лишних файлов и создание необходимых каталогов

Рис. 13.1: Переход в каталог, удаление лишних файлов и создание необходимых каталогов

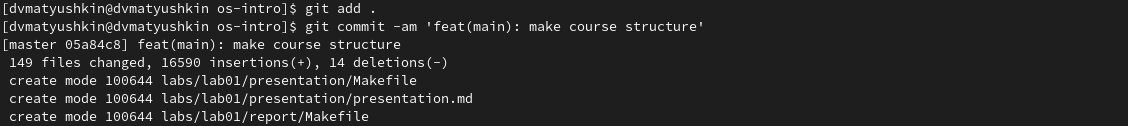


Рис. 13.2: Отправка файлов на сервер

Рис. 13.3: Продолжение вывода

Рис. 13.3: Продолжение вывода

# 3 Заключение:

В ходе этой лабораторной работы мы изучили идеологию и применение средств контроля версий. Освоили умения по работе с git.

# 4 Контрольные вопросы:

1. **Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?** Системы контроля версий (Version Control System, VCS) – система, использующаяся для хранения и управления своим программным продуктом более удобным способом (разработкой). 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.
2. **Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.** Хранилище-система, которая обеспечивает хранение всех существовавших вариантов файлов Commit-фиксация изменений. История-список предыдущих ревизий. Рабочая копия-копия другой ветки. Команде commit можно передать сообщение, описывающее изменения в ревизии. Она также записывает идентификатор пользователя, текущее время и временную зону, плюс список измененных файлов и их содержимого. Сообщение, описывающее изменения, определяется через опцию -m, или – message. Можно также вводить сообщения, состоящие из нескольких строк; в большинстве оболочек вы можете сделать это оставив открытую кавычку в конце строки. commit -m “добавлен первый файл.
3. **Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.** Системы контроля версий. Централизованная система контроля версий Subversion и децентрализованная система контроля версий Mercurial. Существуют СКВ централизованные, в которых имеется один репозиторий, в который собираются изменения со всех рабочих копий разработчиков, и децентрализованные, когда репозиториев много, и они могут обмениваться изменениями между собой. централизованные СКВ - Репозиторий один. У каждого разработчика своя рабочая копия. Время от времени разработчик может затягивать к себе в рабочую копию новые изменения из репозитория, или проталкивать свои изменения из своей рабочей копии в репозиторий. Прочие особенности централизованных СКВ зависят от реализации.
4. **Опишите действия с VCS при единоличной работе с хранилищем.** Создание хранилища (git init). Настройка хранилища. Коммиты нужных файлов (git add , git commit). Пуш файлов (git push). В случае неработоспособности отменить вернуть рабочее состояние исходя из хранимых в хранилище прошлых версий.
5. **Опишите порядок работы с общим хранилищем VCS.** Создание хранилища (git init). Проведение работы локально. Добавление в коммит различных файлов (git add , git commit), коммит ресурсов. Пуш файлов (git push). В случае неработоспособности можно вернуть всё обратно. Если кто-то другой изменил репозиторий - пул ветки.
6. **Каковы основные задачи, решаемые инструментальным средством git?** Предоставление удаленного доступа к коду, хранение всех версий продукта, предоставление возможности отката в случае неработоспособности кода.
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 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.

1. **Приведите примеры использования при работе с локальным и удалённым репозиториями.** Локальные репозитории могут быть полезны когда разработчик один. Удаленные - когда работает целая команда. Так же это может быть и для одного человека - он может иметь доступ к хранилищу удаленно. Как и обычно мы должны создать репозиторий, загрузить его. Тут уже либо у себя, либо удаленно. После этого человек может работать у себя локально и загружать данные на удаленный репозитории. Используя при этом команды commit, add, push , pull и другие
2. **Что такое и зачем могут быть нужны ветви (branches)?** Ветви - отдельные области основного хранилища. Ветви могут быть полезны во многих случаях. Начиная от того, что различные модули программы разрабатываются на одной ветке, заканчивая тем, что ветки могут быть для будущего “код ревью”. Так же могут быть какие-то экспериментальные ветви. Применений множество. Но единый смысл - возможность распараллелить хранилище
3. **Как и зачем можно игнорировать некоторые файлы при commit?** Можно установить специальные настройки в .gitignore. Это может быть полезно, если при работе с файлами могут появляться временные файлы, которые загружать в хранилище нет необходимости.