Отчет по лабораторной работе №2

По дисциплине: Операционные системы

Ситникова Диана Александровна

Москва 2023г

Содержание

# Цель работы

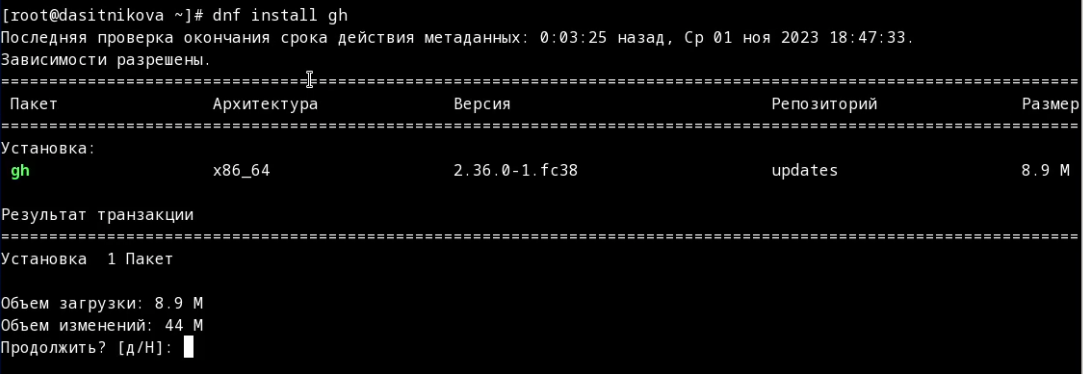
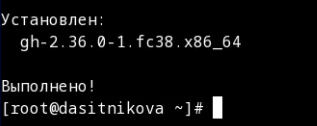
Изучить идеологию и применение средств контроля версий.  
Освоить умения по работе с git.

# Задание

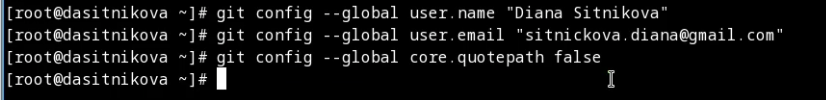
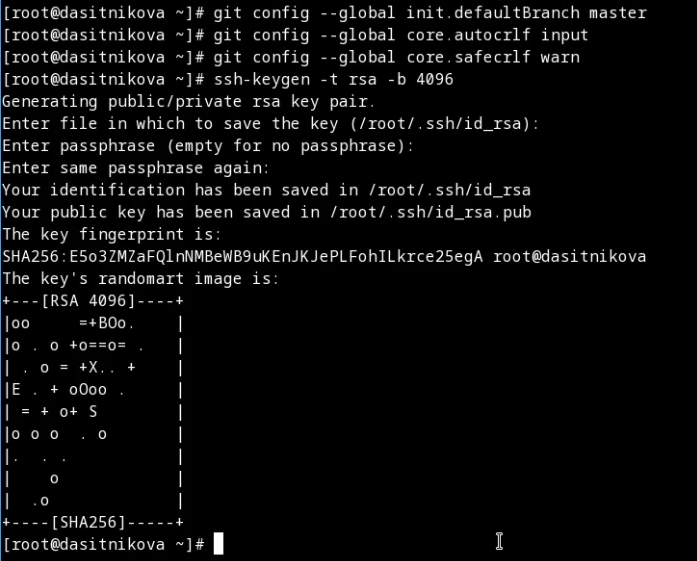
* Создать базовую конфигурацию для работы с git.
* Создать ключ SSH.
* Создать ключ PGP.
* Настроить подписи git.
* Зарегистрироваться на Github.
* Создать локальный каталог для выполнения заданий по предмету.

# Выполнение лабораторной работы

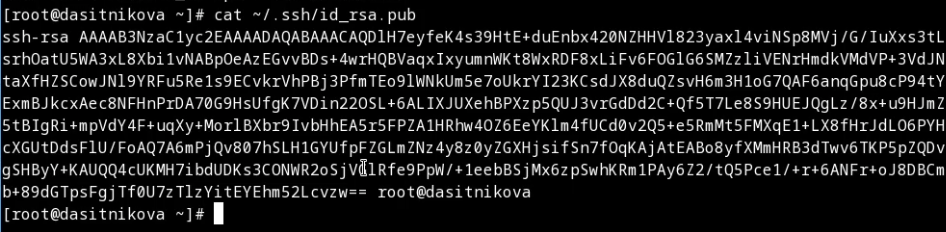
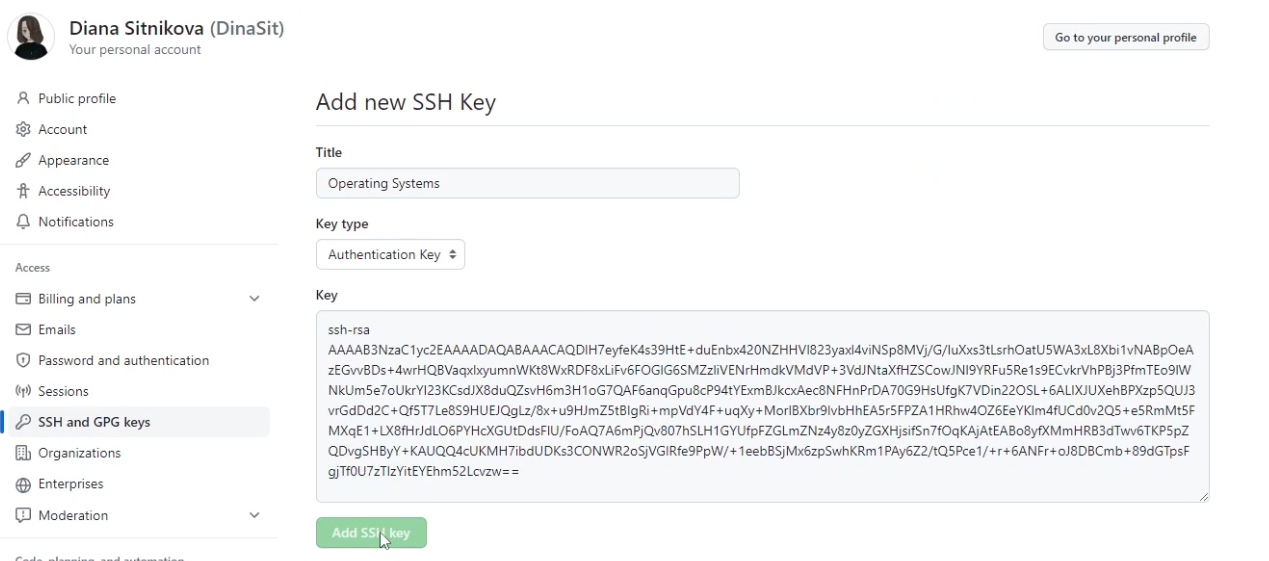
## Установка программного обеспечения

Установим git при помощи команды:  
dnf install git  
  
Установим gh при помощи команды:  
dnf install gh  
  


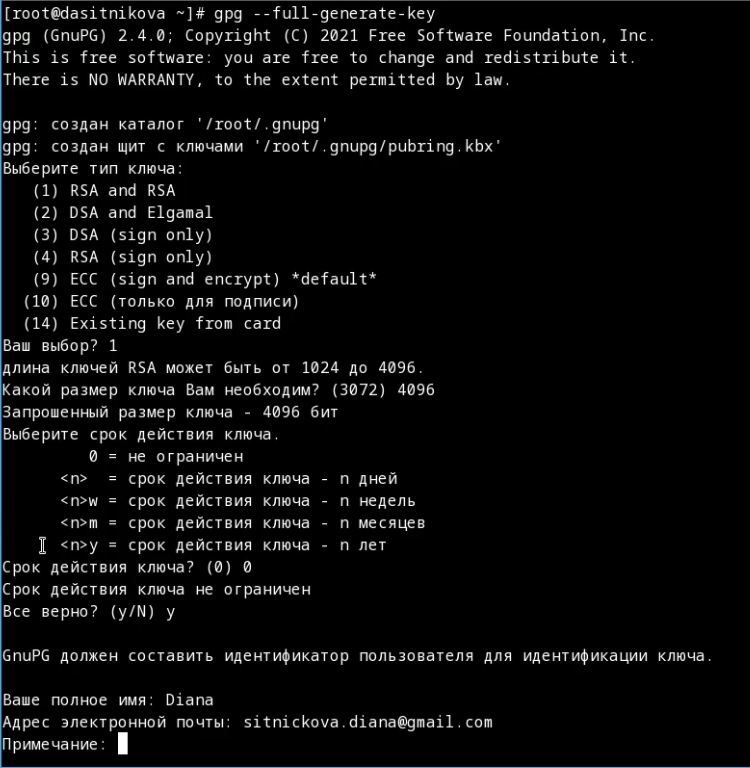
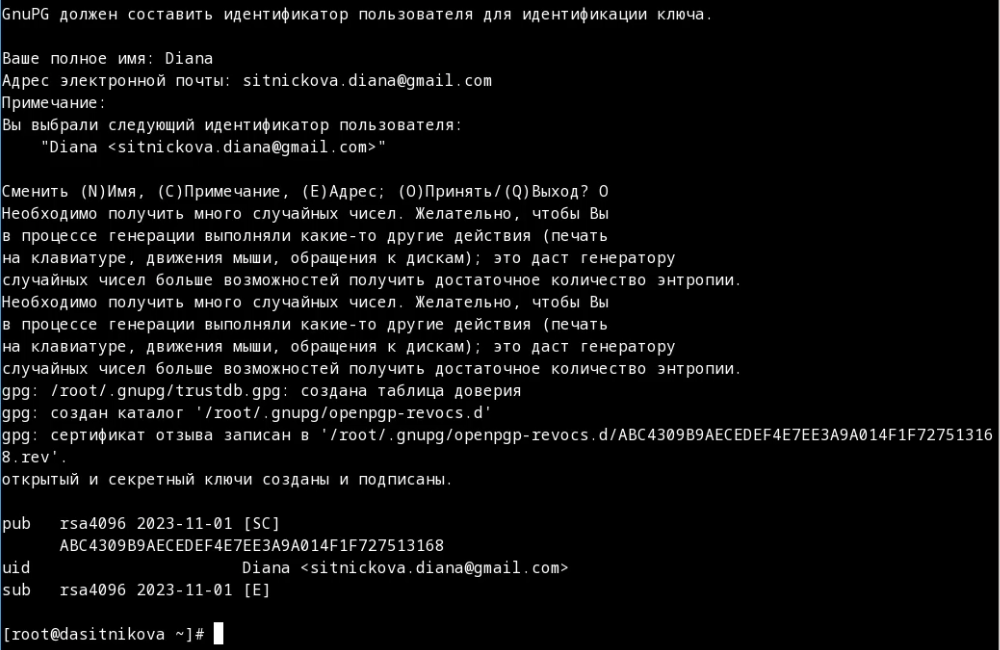
## Базовая настройка git

* Зададим имя и email владельца репозитория при помощи следующих команд:  
  git config –global user.name “Name Surname”  
  git config –global user.email “work@mail”
* Настроим utf-8 в выводе сообщений git командой:  
  git config –global core.quotepath false  
  
* Зададим имя начальной ветки (будем называть её master):  
  git config –global init.defaultBranch master
* Параметр autocrlf:  
  git config –global core.autocrlf input
* Параметр safecrlf:  
  git config –global core.safecrlf warn  
  

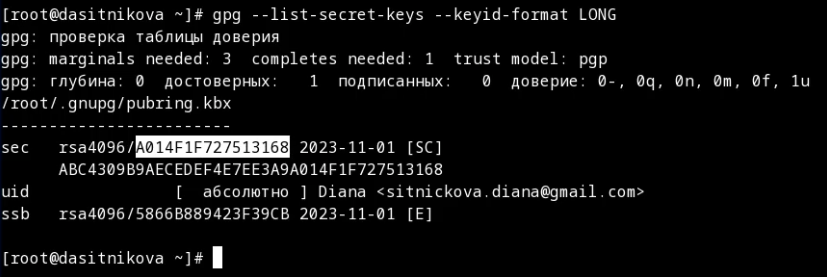
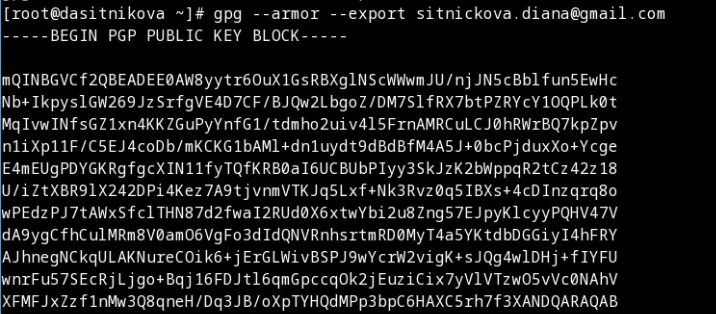
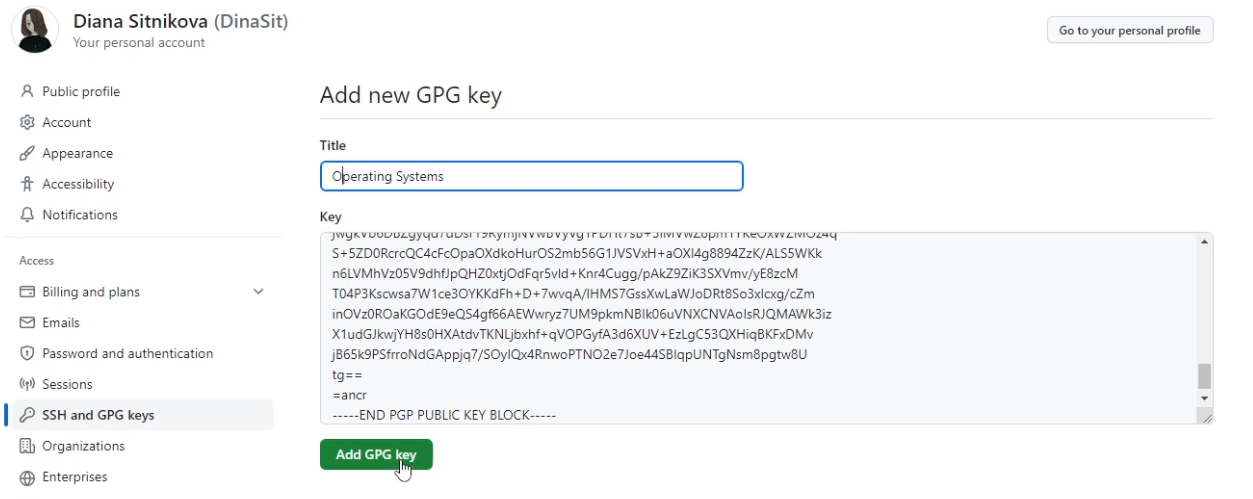
## Созданиче ключа ssh

* по алгоритму rsa с ключём размером 4096 бит:  
  ssh-keygen -t rsa -b 4096  
  (см. “Рис5”)
* для копирования ключа нам понадобится команда:  
  cat ~/.ssh/id\_rsa.pub  
  
* далее добавляем скопированный ключ SSH на GitHub:  
  

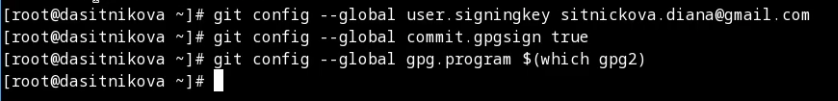
## Создание ключа pgp

* Генерируем ключ при помощи команды:  
  gpg –full-generate-key
* Из предложенных опций выбираем:
  + тип RSA and RSA;
  + размер 4096;
  + выберите срок действия; значение по умолчанию — 0 (срок действия не истекает никогда).
* GPG запросит личную информацию, которая сохранится в ключе:
  + Имя (не менее 5 символов).
  + Адрес электронной почты.
    - При вводе email убедитесь, что он соответствует адресу, используемому на GitHub.
  + Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить это поле пустым.   
    

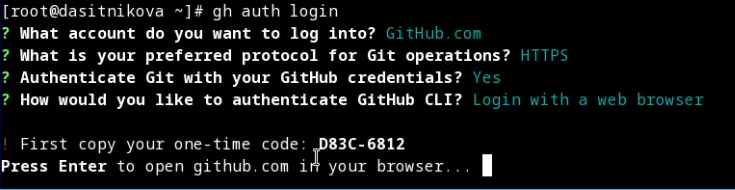
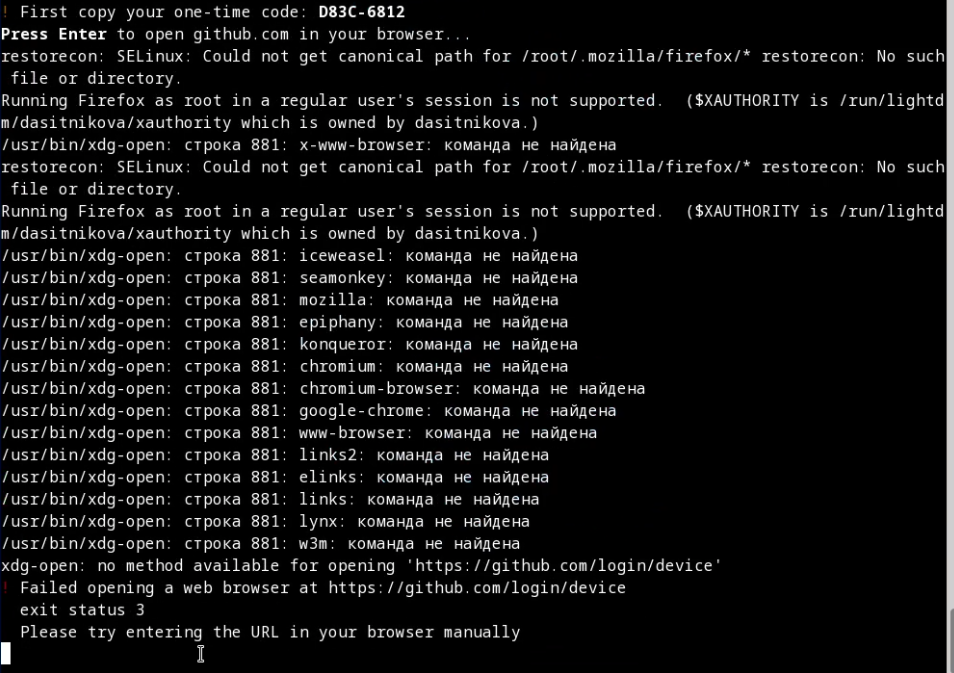
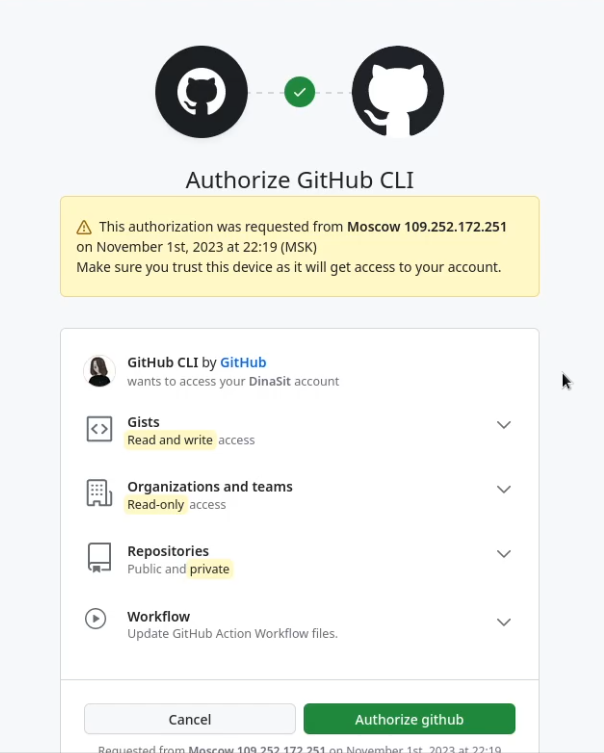
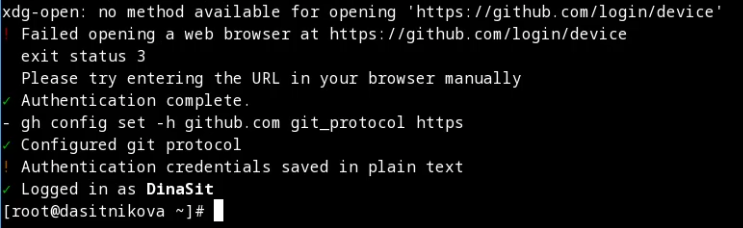
## Добавление PGP ключа в GitHub

* Выводим список ключей и копируем отпечаток приватного ключа: gpg –list-secret-keys –keyid-format LONG
* Отпечаток ключа — это последовательность байтов, используемая для идентификации более длинного, по сравнению с самим отпечатком ключа.
* Формат строки:  
  sec Алгоритм/Отпечаток\_ключа Дата\_создания [Флаги] [Годен\_до] ID\_ключа  
  
* Cкопируйте ваш сгенерированный PGP ключ в буфер обмена:  
  gpg –armor –export | xclip -sel clip   
  Примечание: я использовала в качестве параметра свою почту, указанную при создании ключа PGP, однако, корректнее было бы использовать “Отпечаток\_ключа” как на “Рис10” - A014F1F727513168.
* Перейдите в настройки GitHub (https://github.com/settings/keys), нажмите на кнопку New GPG key и вставьте полученный ключ в поле ввода.  
  

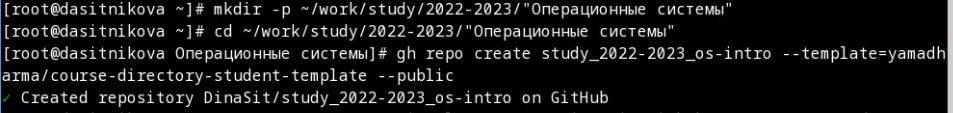
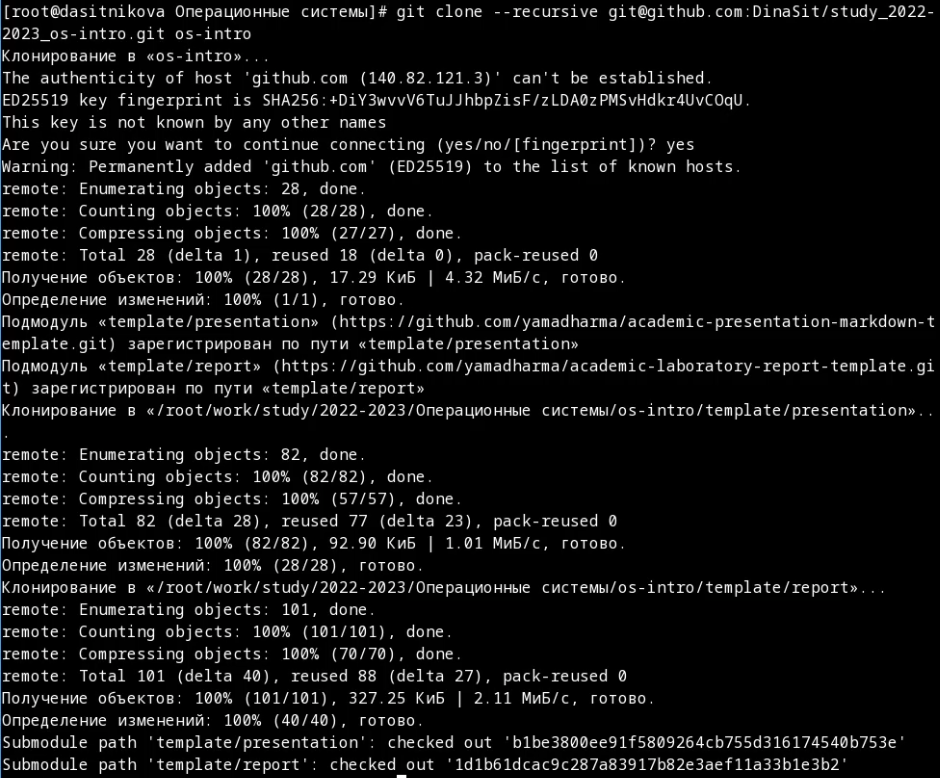
## Настройка автоматических подписей коммитов git

* Используя введёный email, укажите Git применять его при подписи коммитов:  
  git config –global user.signingkey   
  git config –global commit.gpgsign true  
  git config –global gpg.program S(which gpg2)  
  

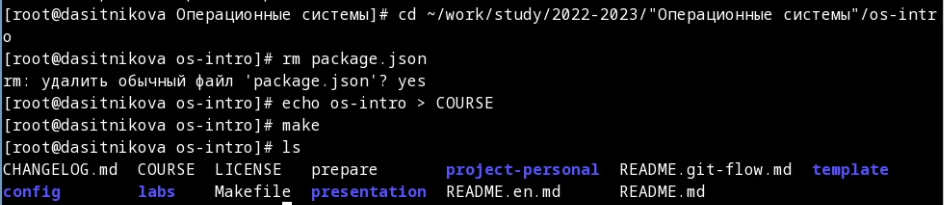
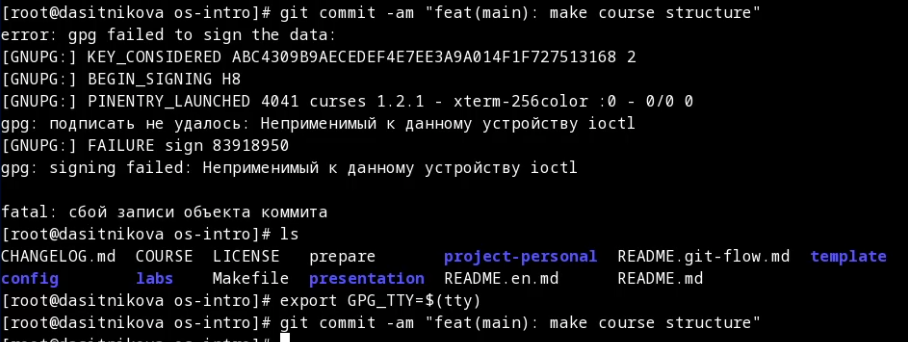
## Настройка gh

* Для начала необходимо авторизоваться  
  gh auth login
* Утилита задаст несколько наводящих вопросов.  
  
* Авторизоваться можно через браузер.   
  
* Результат выполнения корректной работы по настройке gh:  
  

## Сознание репозитория курса на основе шаблона

* Необходимо создать шаблон рабочего пространства (см. Рабочее пространство для лабораторной работы).
* Например, для 2022–2023 учебного года и предмета «Операционные системы» (код предмета os-intro) создание репозитория примет следующий вид:  
  mkdir -p ~/work/study/2022-2023/“Операционные системы”  
  cd ~/work/study/2022-2023/“Операционные системы” gh repo create study\_2022-2023\_os-intro –template=yamadharma/course-directory-student-template –public  
  git clone –recursive git@github.com:/study\_2022-2023\_os-intro.git os-intro  
    
  

## Настройка каталога курса

* Перейдите в каталог курса:  
  cd ~/work/study/2022-2023/“Операционные системы”/os-intro
* Удалите лишние файлы:  
  rm package.json
* Создайте необходимые каталоги:  
  echo os-intro > COURSE  
  make 
* Отправьте файлы на сервер:  
  git add .  
  git commit -am ‘feat(main): make course structure’  
  git push  
    
  Примечание: если при коммите выводит ошибку как на “Рис21”, может помочь команда  
  export GPG\_TTY=$(tty)

# Контрольные вопросы

1. **Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?**  
   Это программное обеспечение для облегчения работы с изменяющейся информацией. VCS 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.
2. **Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.**  
   Хранилище (repository), или репозитарий, — место хранения всех версий и служебной информации.  
   Commit («[трудовой] вклад», не переводится) — синоним версии; процесс создания новой версии.  
   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.
3. **Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.**  
   Централизованные VCS: одно основное хранилище всего проекта и каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет и, затем, добавляет свои изменения обратно. Например: CVS (Concurrent Versions System, Система одновременных версий), Subversion (SVN).  
   Децентрализованные VCS: у каждого пользователя свой вариант (возможно не один) репозитория. Например: Git и Mercurial.
4. **Опишите действия с VCS при единоличной работе с хранилищем.**

* Инициализация репозитория: Создайте новый репозиторий VCS для вашего проекта. В большинстве систем управления версиями это делается командой “init” или подобной.
* Добавление файлов: Добавьте все файлы вашего проекта в репозиторий. Это позволяет начать отслеживать изменения в ваших файлах.
* Создание коммитов: После внесения изменений в файлы сделайте коммиты, чтобы сохранить текущее состояние в репозитории. Коммиты могут включать описание ваших изменений.
* Откат к предыдущим версиям: Если вы сделали изменения, которые вы хотите откатить, VCS позволяет вам вернуться к предыдущим версиям файлов или к более старым коммитам.
* Просмотр истории: VCS сохраняет историю всех ваших коммитов. Вы можете просматривать историю, смотреть, какие изменения были внесены в каждом коммите.
* Ветвление и слияние: В случае необходимости, вы можете создавать ветки (branches) для разработки разных функций или экспериментов. Вы также можете сливать ветки, чтобы объединить изменения из разных веток.
* Работа с удаленным репозиторием: Даже при индивидуальной работе с VCS, вы можете создать удаленный репозиторий (например, на платформе GitHub или GitLab) для резервного копирования и совместного доступа к своему коду с разных устройств.
* Резервное копирование: Регулярно отправляйте свои изменения на удаленный репозиторий или делайте резервное копирование локальных репозиториев для предотвращения потери данных.

1. **Опишите порядок работы с общим хранилищем VCS.**

* Выбор VCS и настройка: В начале проекта выберите VCS, который лучше всего соответствует вашим потребностям. Популярными системами управления версиями являются Git, SVN, Mercurial, и другие. Установите и настройте выбранную систему на вашем компьютере.
* Инициализация репозитория: Создайте новый репозиторий (хранилище) VCS для вашего проекта. Это можно сделать с помощью команды init или аналогичной в зависимости от выбранной системы.
* Добавление файлов: Добавьте все файлы вашего проекта в локальный репозиторий. В Git, это делается с помощью команды git add, в SVN - команды svn add, и так далее.
* Создание коммитов: После добавления файлов, создайте коммиты (snapshots) для сохранения текущего состояния вашего проекта в локальном репозитории. Коммиты должны сопровождаться описанием ваших изменений. В Git, коммиты создаются с помощью команды git commit, в SVN - svn commit.
* Работа с удаленным репозиторием: Если у вас есть общий проект с другими разработчиками, существует удаленный репозиторий на сервере. Вы можете клонировать (создать локальную копию) этого удаленного репозитория с помощью команды git clone, svn checkout, и других в зависимости от системы VCS.
* Работа над проектом: Вы и другие разработчики можете работать над проектом, внося изменения в файлы в вашей локальной копии. По мере работы, регулярно создавайте коммиты, чтобы сохранить изменения в вашем локальном репозитории.
* Отправка изменений: Когда вы готовы поделиться своими изменениями с другими участниками проекта, отправьте их на удаленный репозиторий с помощью команды git push, svn commit, или аналогичных команд в других системах.
* Обновление локальной копии: Для получения изменений, внесенных другими участниками проекта, используйте команду обновления (git pull, svn update и др.).
* Работа с ветками: При необходимости, создавайте и работайте с ветками (branches) для разработки разных функций. Ветвление и слияние (merge) - важные аспекты работы с VCS.
* Решение конфликтов: Если возникают конфликты при слиянии изменений, решайте их вручную и сохраняйте изменения в конфликтных файлах.
* Отслеживание истории: VCS сохраняет историю всех изменений в проекте. Вы можете просматривать историю, анализировать изменения и возвращаться к предыдущим версиям при необходимости.
* Резервное копирование: Регулярно сохраняйте резервные копии вашего локального репозитория и удаленного репозитория, чтобы избежать потери данных.

1. **Каковы основные задачи, решаемые инструментальным средством git?**  
   Git — это система управления версиями. У Git две основных задачи:

* хранить информацию о всех изменениях в вашем коде, начиная с самой первой строчки
* 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 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

1. **Приведите примеры использования при работе с локальным и удалённым репозиториями.**  
   Работа с локальным репозиторием:

* git init: Инициализация нового локального репозитория.
* git clone : Клонирование удаленного репозитория в локальную директорию.
* git status: Показ текущего состояния локального репозитория, включая измененные, добавленные и неотслеживаемые файлы.
* git add : Добавление файла в индекс, подготовка к коммиту.
* git commit -m “Сообщение коммита”: Создание коммита с сообщением, описывающим внесенные изменения.
* git log: Просмотр истории коммитов в локальном репозитории.
* git branch: Просмотр списка веток в репозитории.
* git checkout : Переключение на другую ветку.
* git merge : Слияние изменений из указанной ветки в текущую.
* git reset : Откат на определенный коммит, отмена изменений.  
  Работа с удаленным репозиторием:
* git remote -v: Просмотр списка удаленных репозиториев, связанных с текущим локальным репозиторием.
* git pull : Получение изменений с удаленного репозитория и объединение их с текущей веткой.
* git push : Отправка своих локальных изменений на удаленный репозиторий.
* git fetch : Получение информации о состоянии удаленного репозитория без слияния изменений.
* git clone : Клонирование удаленного репозитория в локальную директорию.

1. **Что такое и зачем могут быть нужны ветви (branches)?**  
   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.
2. **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.

# Выводы

В ходе выполнения лабораторной работы я изучила идеологию и применение средств контроля версий, а также освоила умения по работе с git.