

# **Лабораторная работа 2**

**Управление версиями**

Парфенова Елизавета Евгеньевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>11</b>
<b>5</b>	<b>Выводы</b>	<b>21</b>
<b>6</b>	<b>Контрольные вопросы</b>	<b>22</b>

## Список иллюстраций

4.1	Регистрация на GitHub . . . . .	11
4.2	Настройка основных данных аккаунта . . . . .	12
4.3	Установка git-flow . . . . .	12
4.4	Установка gh . . . . .	13
4.5	Базовая настройка git . . . . .	13
4.6	Создание ключа ssh . . . . .	14
4.7	Создание ключа ssh . . . . .	14
4.8	Создание ключа pgr . . . . .	15
4.9	Отпечаток приватного ключа . . . . .	16
4.10	Копирование ключа в буфер обмена . . . . .	16
4.11	Добавление ключа в Github . . . . .	16
4.12	Настройка автоматических подписей коммитов git . . . . .	17
4.13	Настройка gh . . . . .	17
4.14	Копирование ключа в буфер обмена . . . . .	17
4.15	Добавление ключа в Github . . . . .	18
4.16	Создание репозитория курса . . . . .	18
4.17	Создание репозитория курса . . . . .	19
4.18	Настройка каталога курса . . . . .	20
4.19	Настройка каталога курса . . . . .	20

## **Список таблиц**

# 1 Цель работы

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

## 2 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.

## 3 Теоретическое введение

### **Системы контроля версий. Общие понятия**

*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.

### **Основные команды git**

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***

- 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***

- 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(при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой):

### ***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 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***

## **Работа с локальным репозиторием**

Создадим локальный репозиторий.

Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория:

***git config --global user.name "Имя Фамилия"***

***git config --global user.email "work@mail"***

и настроив utf-8 в выводе сообщений git:

***git config --global core.quotePath false***

Для инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial, необходимо ввести в командной строке:

***cd***

***mkdir tutorial***

***cd tutorial***

***git init***

После это в каталоге tutorial появится каталог .git, в котором будет храниться история изменений.

Создадим тестовый текстовый файл hello.txt и добавим его в локальный репозиторий:

***echo 'hello world' > hello.txt git add hello.txt git commit -am 'Новый файл'***

Воспользуемся командой status для просмотра изменений в рабочем каталоге, сделанных с момента последней ревизии:

***git status***

## 4 Выполнение лабораторной работы

Вначале я перешла по ссылке <https://github.com> и создала учетную запись на github. (рис. 4.1)

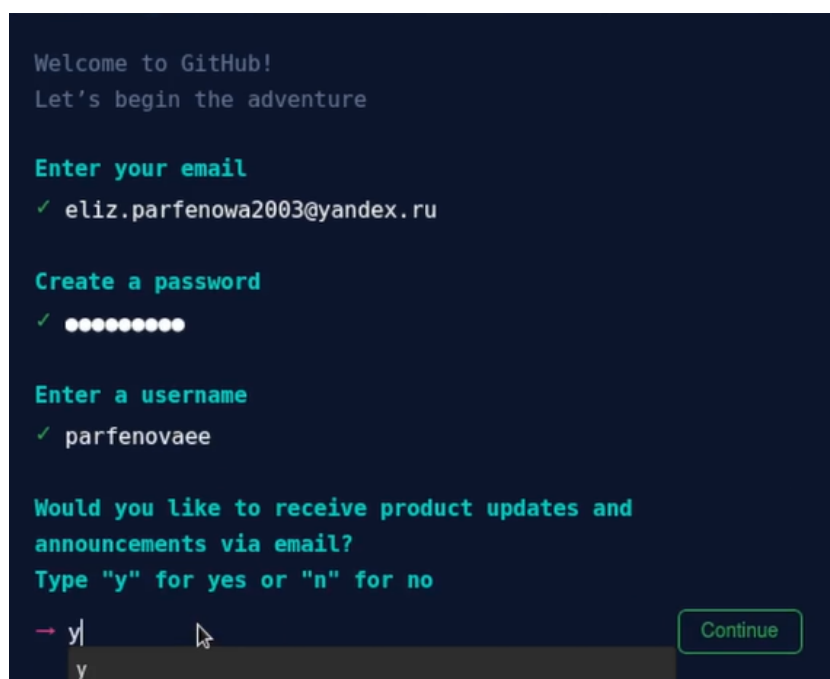


Рис. 4.1: Регистрация на GitHub

После подтверждения кодом, который пришел на электронную почту, я настроила основные данные аккаунта. (рис. 4.2)

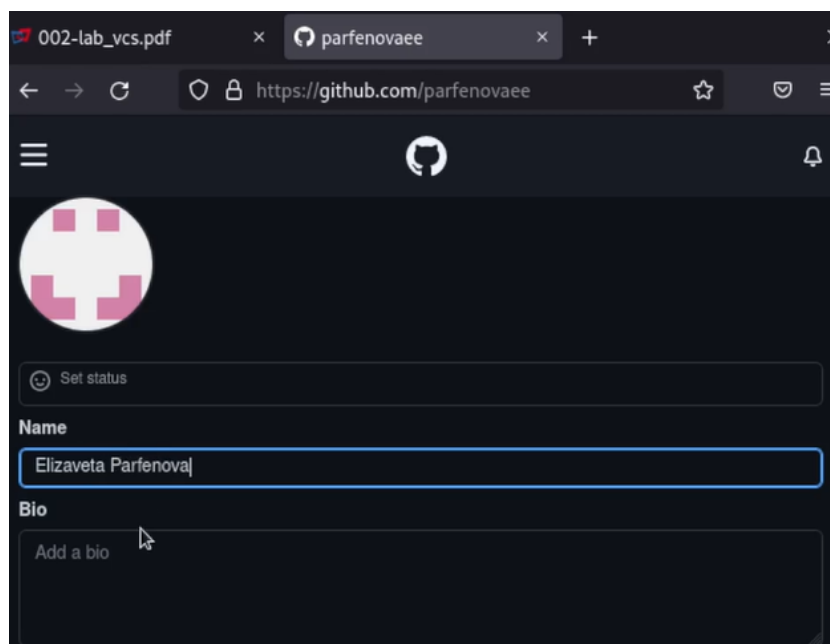


Рис. 4.2: Настройка основных данных аккаунта

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.

Команды:

- ***wget --no-check-certificate -q https://raw.githubusercontent.com/petervanderdoes/gitflow/develop/contrib/gitflow-installer.sh***
- ***chmod +x gitflow-installer.sh***
- ***sudo ./gitflow-installer.sh install stable*** (рис. 4.3)

```
[eeeparfenova@fedora ~]$ cd /tmp
[eeeparfenova@fedora tmp]$ wget --no-check-certificate -q https://raw.githubusercontent.com/petervanderdoes/gitflow/develop/contrib/gitflow-installer.sh
[eeeparfenova@fedora tmp]$ chmod +x gitflow-installer.sh
[eeeparfenova@fedora tmp]$ sudo ./gitflow-installer.sh install stable
[sudo] пароль для eeeparfenova:
## git-flow no-make installer ##
Installing git-flow to /usr/local/bin
Cloning repo from GitHub to gitflow
Клонирование в «gitflow»...
remote: Enumerating objects: 4270, done.
remote: Total 4270 (delta 0), reused 0 (delta 0), pack-reused 4270
Получение объектов: 100% (4270/4270), 1.74 МБ | 363.00 КиБ/с, готово.
Определение изменений: 100% (2533/2533), готово.
```

Рис. 4.3: Установка git-flow

Далее я провела установку gh в Fedora Linux с помощью команды ***sudo dnf***

**install gh.** (рис. 4.4)

```
[eeeparfenova@fedora tmp]$ sudo dnf install gh
Fedora 35 - x86_64 - Updates          1.7 kB/s | 7.4 kB      00:04
Fedora 35 - x86_64 - Updates          81 kB/s | 2.6 MB      00:32
Fedora Modular 35 - x86_64 - Updates  32 kB/s | 19 kB       00:00
Пакет gh-2.7.0-1.fc35.x86_64 уже установлен.
Зависимости разрешены.
Отсутствуют действия для выполнения.
Выполнено!
```

Рис. 4.4: Установка gh

Следующим шагом я совершила базовую настройку git.

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. 4.5)

```
[eeeparfenova@fedora tmp]$ git config --global user.name "Elizaveta Parfenova"
[eeeparfenova@fedora tmp]$ git config --global user.email "eliz.parfenowa2003@yandex.ru"
[eeeparfenova@fedora tmp]$ git config --global core.quotepath false
[eeeparfenova@fedora tmp]$ git config --global init.defaultBranch master
[eeeparfenova@fedora tmp]$ git config --global core.autocrlf input
[eeeparfenova@fedora tmp]$ git config --global core.safecrlf warn
```

Рис. 4.5: Базовая настройка git

Следующий шаг – создание ключа ssh. Вначале командой **ssh-keygen -t rsa -b 4096** я создаю ключ по алгоритму rsa размером 4096 бит, а после командой **ssh-keygen -t ed25519** – по алгоритму ed25519. (рис. 4.6) (рис. 4.7)

```
[eeeparfenova@fedora tmp]$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/eeeparfenova/.ssh/id_rsa):
/home/eeeparfenova/.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/eeeparfenova/.ssh/id_rsa
Your public key has been saved in /home/eeeparfenova/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:YCZx4Lr3YkPJLmwlp0JziGLF9kpkwsBpACl5zB+0e00 eeeparfenova@fedora
The key's randomart image is:
+---[RSA 4096]-----+
|+= .o                |
|* =0.+               |
|0=..=.o              |
|+ o.= .              |
| +0+..E S            |
|. B==o               |
|oo=Bo                |
|o=o=o=o              |
|o.o+.o.              |
+---[SHA256]-----+
```




Рис. 4.6: Создание ключа ssh

```
[eeeparfenova@fedora tmp]$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/eeeparfenova/.ssh/id_ed25519):
/home/eeeparfenova/.ssh/id_ed25519 already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/eeeparfenova/.ssh/id_ed25519
Your public key has been saved in /home/eeeparfenova/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:eYF69WpMB0LLWmj88Z/ldN7z9GSmAaKd/ges/WqXf4 eeeparfenova@fedora
The key's randomart image is:
+--[ED25519 256]--+
|
| .
| = o .
| B O +
| + = * +
| + S o o o
| * + . o+.o|
| . B + =. +oo|
| o + B.+ +o|
| .*oo.o.E|
+---[SHA256]-----+
```




Рис. 4.7: Создание ключа ssh

Следующим шагом я создала ключ pgr. 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. 4.8)

```
[eeeparfenova@fedora tmp]$ gpg --full-generate-key
gpg (GnuPG) 2.3.4; 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 and RSA
(2) DSA and Elgamal
(3) DSA (sign only)
(4) RSA (sign only)
(9) ECC (sign and encrypt) *default*
(10) ECC (только для подписи)
(14) Existing key from card
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: Elizaveta Parfenova
Адрес электронной почты: eliz.parfenowa2003@yandex.ru
Примечание:
Вы выбрали следующий идентификатор пользователя:
  "Elizaveta Parfenova <eliz.parfenowa2003@yandex.ru>"
```

Рис. 4.8: Создание ключа gpg

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. 4.9)

```
[eeparfenova@fedora tmp]$ gpg --list-secret-keys --keyid-format LONG
gpg: проверка таблицы доверия
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: глубина: 0 достоверных: 2 подписанных: 0 доверие: 0-, 0q, 0n, 0m, 0f, 2
u
/home/eeparfenova/.gnupg/pubring.kbx
-----
sec   rsa4096/B124890584117CED 2022-04-21 [SC]
      91AEF6D65F84AF6E4653ABAFB124890584117CED
uid           [ абсолютно ] Elizabeth Parfenova <eliz.parfenowa2003@yandex.ru>
ssb   rsa4096/55190313E3005F73 2022-04-21 [E]

sec   rsa4096/B60AC448BC125F11 2022-04-22 [SC]
      BC524E186C1B2B83C4A73C31B60AC448BC125F11
uid           [ абсолютно ] Elizaveta Parfenova <eliz.parfenowa2003@yandex.ru>
ssb   rsa4096/27A5AEF72A237BF4 2022-04-22 [E]
```

Рис. 4.9: Отпечаток приватного ключа

Скопировала его и вставила в данную команду ***gpg --armor --export | xclip -sel clip вместо*** . Это нужно было для того, чтобы скопировать ключ в буфер обмена. (рис. 4.10)

```
[eeparfenova@fedora tmp]$ gpg --armor --export B124890584117CED | xclip -sel clip
```

Рис. 4.10: Копирование ключа в буфер обмена

После я зашла на Github и добавила ключ туда, во вкладку GPG Keys. (рис. 4.11)

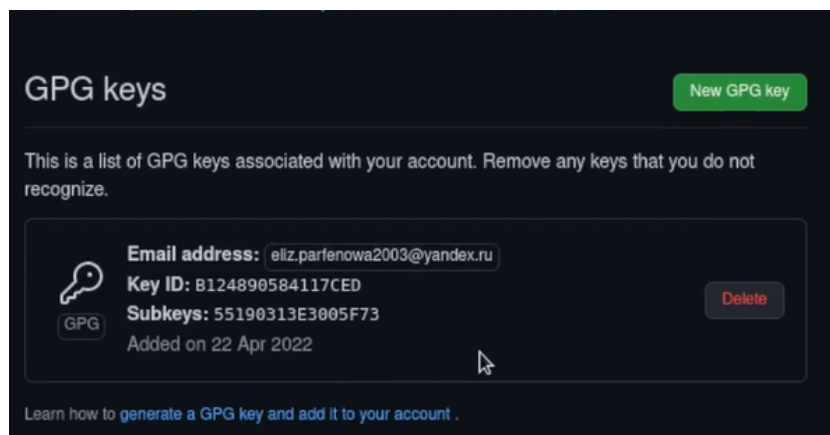


Рис. 4.11: Добавление ключа в Github

Далее я настроила автоматические подписи коммитов git. Для этого используем введенный email. (рис. 4.12)



```
[eeeparfenova@fedora tmp]$ gpg --armor --export B124890584117CED | xclip -sel clip
[eeeparfenova@fedora tmp]$ git config --global user.signingkey B124890584117CED
[eeeparfenova@fedora tmp]$ git config --global commit.gpgsign true
[eeeparfenova@fedora tmp]$ git config --global gpg.program $(which gpg2)
```

Рис. 4.12: 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. 4.13)

```
[eeeparfenova@fedora tmp]$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? HTTPS
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: 2FA9-DEBE
Press Enter to open github.com in your browser...
✓ Authentication complete.
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as parfenovae
```

Рис. 4.13: 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: parfenovae/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. 4.14)

```
[eeeparfenova@fedora Операционные системы]$ cat ~/.ssh/id_rsa.pub | xclip -sel clip
```

Рис. 4.14: Копирование ключа в буфер обмена

После переходим на Github в уже знакомую у , где добавляли GPG ключ, и добавляем SSH ключ в нужное окно. (рис. 4.15)

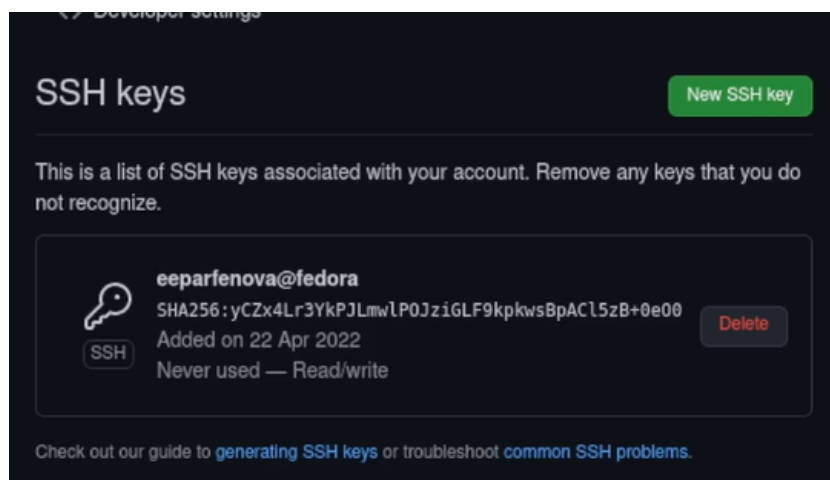


Рис. 4.15: Добавление ключа в Github

Теперь все команды по созданию репозитория курса могут быть выполнены успешно.(рис. 4.16) (рис. 4.17)

```
[eeparfenova@fedora tmp]$ mkdir -p ~/work/study/2021-2022/"Операционные системы"
[eeparfenova@fedora tmp]$ cd ~/work/study/2021-2022/"Операционные системы"
[eeparfenova@fedora Операционные системы]$ gh repo create study_2021-2022_os-intro-
--template=yamadharma/course-directory-student-template --public
HTTP 404: Not Found (https://api.github.com/users/study_2021-2022_os-intro--templat
e=yamadharma)
[eeparfenova@fedora Операционные системы]$ gh repo create study_2021-2022_os-intro
--template=yamadharma/course-directory-student-template --public
✓ Created repository parfenovae/study_2021-2022_os-intro on GitHub
```

Рис. 4.16: Создание репозитория курса

```
[eeparfenova@fedora Операционные системы]$ git clone --recursive git@github.com:parfenovae/study_2021-2022_os-intro.git os-intro
Клонирование в «os-intro»...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 20 (delta 2), reused 15 (delta 2), pack-reused 0
Получение объектов: 100% (20/20), 12.50 КиБ | 6.25 МиБ/с, готово.
Определение изменений: 100% (2/2), готово.
Подмодуль «template/presentation» (https://github.com/yamadharma/academic-presentation-markdown-template.git) зарегистрирован по пути «template/presentation»
Подмодуль «template/report» (https://github.com/yamadharma/academic-laboratory-report-template.git) зарегистрирован по пути «template/report»
Клонирование в «/home/eeparfenova/work/study/2021-2022/Операционные системы/os-intro/template/presentation»...
remote: Enumerating objects: 42, done.
remote: Counting objects: 100% (42/42), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 42 (delta 9), reused 40 (delta 7), pack-reused 0
Получение объектов: 100% (42/42), 31.19 КиБ | 3.12 МиБ/с, готово.
Определение изменений: 100% (9/9), готово.
Клонирование в «/home/eeparfenova/work/study/2021-2022/Операционные системы/os-intro/template/report»...
remote: Enumerating objects: 78, done.
remote: Counting objects: 100% (78/78), done.
remote: Compressing objects: 100% (52/52), done.
remote: Total 78 (delta 31), reused 69 (delta 22), pack-reused 0
Получение объектов: 100% (78/78), 292.27 КиБ | 587.00 КиБ/с, готово.
Определение изменений: 100% (31/31), готово.
Submodule path 'template/presentation': checked out '3eae6b7586f8a9aded2b506cd1018e625b228b93'
Submodule path 'template/report': checked out 'df7b2ef80f8def3b9a496f8695277469a1a7842a'
```

Рис. 4.17: Создание репозитория курса

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 :

- **git add**.
- **git commit -am 'feat(main): make course structure'** (эта команда требует кодовой фразы, которую мы вводили ранее)
- **git push** (рис. 4.18) (рис. 4.19)

```

[eeeparfenova@fedora Операционные системы]$ cd ~/work/study/2021-2022/"Операционные
системы"/os-intro
[eeeparfenova@fedora os-intro]$ rm package.json
[eeeparfenova@fedora os-intro]$ make COURSE=os-intro
[eeeparfenova@fedora os-intro]$ git add .
[eeeparfenova@fedora os-intro]$ git commit -am 'feat(main): make course structure'
error: gpg не удалось подписать данные
fatal: сбой записи объекта коммита
[eeeparfenova@fedora os-intro]$ git commit -am 'feat(main): make course structure'
[master d04b413] feat(main): make course structure
149 files changed, 16590 insertions(+), 14 deletions(-)

```

Рис. 4.18: Настройка каталога курса

```

[eeeparfenova@fedora os-intro]$ git push
Перечисление объектов: 20, готово.
Подсчет объектов: 100% (20/20), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (16/16), готово.
Запись объектов: 100% (19/19), 266.53 КиБ | 2.18 МБ/с, готово.
Всего 19 (изменений 2), повторно использовано 0 (изменений 0), повторно использован
о пакетов 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To github.com:parfenovae/study_2021-2022_os-intro.git
d7ab0bb..d04b413 master -> master

```

Рис. 4.19: Настройка каталога курса

## 5 Выводы

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

## 6 Контрольные вопросы

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

Система контроля версий (VCS) - программное обеспечение 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 разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов.

2. 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.

3. 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 репозитория. Пример – Git.

4. 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.

5. 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.

6. 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.

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 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` – 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: `git push origin имя_ветки` – слияние ветки с текущим деревом: `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`

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

Работа с удалённым репозиторием: `git remote` – просмотр списка настроенных удалённых репозиториях.

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

9. 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.

10. 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`