

# **Отчёт по лабораторной работе №2**

**Первоначальная настройка git**

Касакьянц Владислав Сергеевич

# 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>8</b>
<b>5</b>	<b>Контрольные вопросы</b>	<b>16</b>
<b>6</b>	<b>Выводы</b>	<b>21</b>
	<b>Список литературы</b>	<b>22</b>

## Список иллюстраций

4.1	Установка git и gh . . . . .	8
4.2	Базовая настройка git . . . . .	9
4.3	Создание ssh ключа . . . . .	9
4.4	Создание pgr ключа . . . . .	10
4.5	Учетная запись GitHub . . . . .	10
4.6	Копирование pgr ключа . . . . .	11
4.7	Добавление ключей . . . . .	11
4.8	Настройка коммитов . . . . .	11
4.9	Авторизация . . . . .	12
4.10	Авторизация . . . . .	12
4.11	Создание репозитория . . . . .	13
4.12	Настройка каталога курса . . . . .	14
4.13	Отправка на сервер . . . . .	15

## Список таблиц

# 1 Цель работы

Целью данной работы заключается в изучении идеологии и применении средств контроля версий, а также освоить умения по работе с git.

## 2 Задание

1. Создать базовую конфигурацию для работы с git.
2. Создать ключ SSH.
3. Создать ключ PGP.
4. Настроить подписи git.
5. Зарегистрироваться на Github.
6. Создать локальный каталог для выполнения заданий по предмету.

### 3 Теоретическое введение

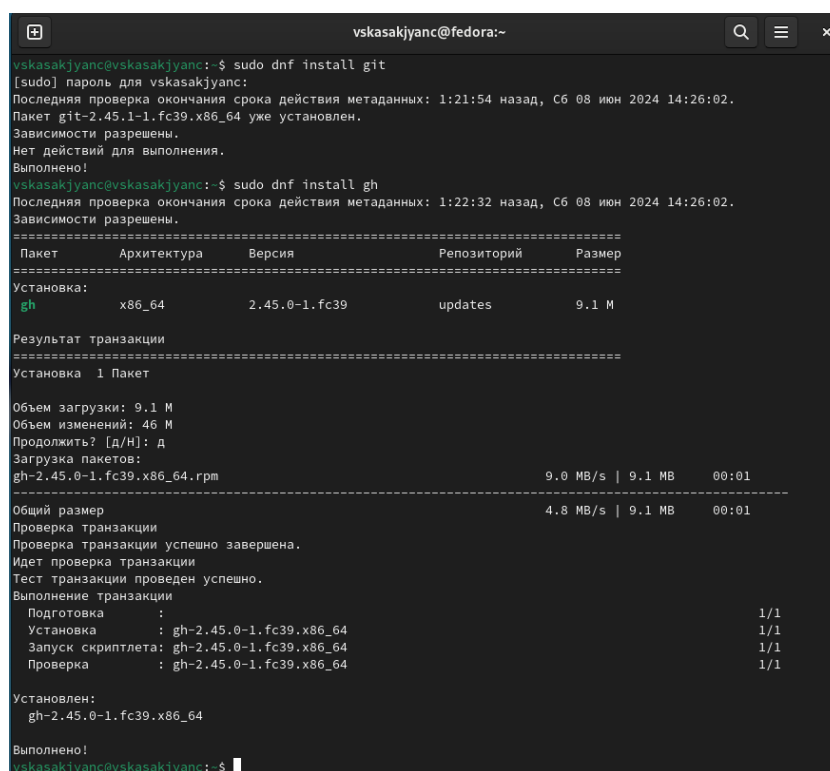
Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется.

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.

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.

## 4 Выполнение лабораторной работы

Для установки **git** вводим команду `sudo dnf install git`. Должна пойти установка, но у меня **git** уже установлен. Так же устанавливаем **gh**, введя команду `dnf install gh` (рис. 4.1).



```
vskasakjyanc@fedora:~$ sudo dnf install git
[sudo] пароль для vskasakjyanc:
Последняя проверка окончания срока действия метаданных: 1:21:54 назад, Сб 08 июн 2024 14:26:02.
Пакет git-2.45.1-1.fc39.x86_64 уже установлен.
Зависимости разрешены.
Нет действий для выполнения.
Выполнено!

vskasakjyanc@vskasakjyanc:~$ sudo dnf install gh
Последняя проверка окончания срока действия метаданных: 1:22:32 назад, Сб 08 июн 2024 14:26:02.
Зависимости разрешены.
=====
Пакет      Архитектура  Версия      Репозиторий  Размер
=====
Установка:
  gh        x86_64       2.45.0-1.fc39  updates      9.1 M
=====
Результат транзакции
=====
Установка 1 Пакет

Объем загрузки: 9.1 M
Объем изменений: 46 M
Продолжить? [д/н]: д
Загрузка пакетов:
gh-2.45.0-1.fc39.x86_64.rpm                      9.0 MB/s | 9.1 MB   00:01
-----
Общий размер                                     4.8 MB/s | 9.1 MB   00:01
Проверка транзакции
Проверка транзакции успешно завершена.
Идет проверка транзакции
Тест транзакции проведен успешно.
Выполнение транзакции
  Подготовка      :
  Установка       : gh-2.45.0-1.fc39.x86_64      1/1
  Запуск скрипта  : gh-2.45.0-1.fc39.x86_64      1/1
  Проверка        : gh-2.45.0-1.fc39.x86_64      1/1
Установлен:
gh-2.45.0-1.fc39.x86_64
Выполнено!
vskasakjyanc@vskasakjyanc:~$
```

Рис. 4.1: Установка git и gh

Сделаем базовые настройки **git**. Для этого зададим имя и почту владельца репозитория, настроим utf-8 в выводе сообщений **git**, зададим имя начальной ветки, которую будем называть ее **master** и установим пару параметров. И проверим изменения с помощью команды `git config --list` (рис. 4.2).



```
vskasakjyanc@fedora:~$ git config --global user.name "vskasakjyanc"
vskasakjyanc@vskasakjyanc:~$ git config --global user.email "golod99@mail.ru"
vskasakjyanc@vskasakjyanc:~$ git config --global core.quotepath false
vskasakjyanc@vskasakjyanc:~$ git config --global init.defaultBranch master
vskasakjyanc@vskasakjyanc:~$ git config --global core.autocrlf input
vskasakjyanc@vskasakjyanc:~$ git config --global core.safecrlf warn
vskasakjyanc@vskasakjyanc:~$ git config --list
user.name=vskasakjyanc
user.email=golod99@mail.ru
core.quotepath=false
core.autocrlf=input
core.safecrlf=warn
init.defaultbranch=master
vskasakjyanc@vskasakjyanc:~$
```

Рис. 4.2: Базовая настройка git

Далее создадим ключ **ssh** по алгоритму **rsa** с ключом размером 4096 бит с помощью команды `ssh-keygen -t rsa -b 4096` (рис. 4.3).

```
vskasakjyanc@fedora:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vskasakjyanc/.ssh/id_rsa):
Created directory '/home/vskasakjyanc/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vskasakjyanc/.ssh/id_rsa
Your public key has been saved in /home/vskasakjyanc/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:yhKln0sLmPhG1YyZiFfZqUW87VPzoBzc6+/bShlrHs vskasakjyanc@vskasakjyanc
The key's randomart image is:
+----[RSA 4096]-----+
|  .  .  .  |
|  .  .B =  |
|  .  . = +  |
|  .  . + =  |
|  .  .o$+ =  |
|o.  + . = . . o|
|=.  o o o . + |
|o o . . . o.E |
|  o.  ++oo+. |
+----[SHA256]-----+
vskasakjyanc@vskasakjyanc:~$
```

Рис. 4.3: Создание ssh ключа

Дальше сгенерируем ключ **pgp** с помощью команды `gpg --full-generate-key`.

Из предложенных опций выбираем тип **RSA** и **RSA**, размер 4096 и срок действия 0 (срок действия не истекает никогда). Так же вводим личную информацию, которая сохранится в ключе (рис. 4.4).

```
vskasakjyanc@fedora:~$ gpg --full-generate-key
gpg (GnuPG) 2.4.4; Copyright (C) 2024 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: создан каталог '/home/vskasakjyanc/.gnupg'
Выберите тип ключа:
  (1) RSA and RSA
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
  (5) 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: vskasakjyanc
Адрес электронной почты: golod99@mail.ru
Примечание:
Вы выбрали следующий идентификатор пользователя:
"vskasakjyanc <golod99@mail.ru>"

Сменить (N) имя, (C) Примечание, (E) Адрес; (O) Принять/(Q) Выход? 0
Необходимо получить много случайных чисел. Желательно, чтобы Вы
в процессе генерации выполняли какие-то другие действия (печать
на клавиатуре, движения мыши, обращения к диску); это даст генератору
случайных чисел больше возможностей получить достаточное количество энтропии.
Необходимо получить много случайных чисел. Желательно, чтобы Вы
в процессе генерации выполняли какие-то другие действия (печать
на клавиатуре, движения мыши, обращения к диску); это даст генератору
случайных чисел больше возможностей получить достаточное количество энтропии.
gpg: /home/vskasakjyanc/.gnupg/trustdb.gpg: создана таблица доверия
gpg: создан каталог '/home/vskasakjyanc/.gnupg/openpgp-revocs.d'
gpg: сертификат отзыва записан в '/home/vskasakjyanc/.gnupg/openpgp-revocs.d/2C9B261D1B28110A4685FFE4C83527CCFFC7BDD4
.rev'.
открытый и секретный ключи созданы и подписаны.

pub   rsa4096 2024-06-08 [SC]
      2C9B261D1B28110A4685FFE4C83527CCFFC7BDD4
uid    vskasakjyanc <golod99@mail.ru>
sub    rsa4096 2024-06-08 [E]

vskasakjyanc@vskasakjyanc:~$
```

Рис. 4.4: Создание ргр ключа

После создадим учетную запись на GitHub, но у меня уже есть (рис. 4.5).

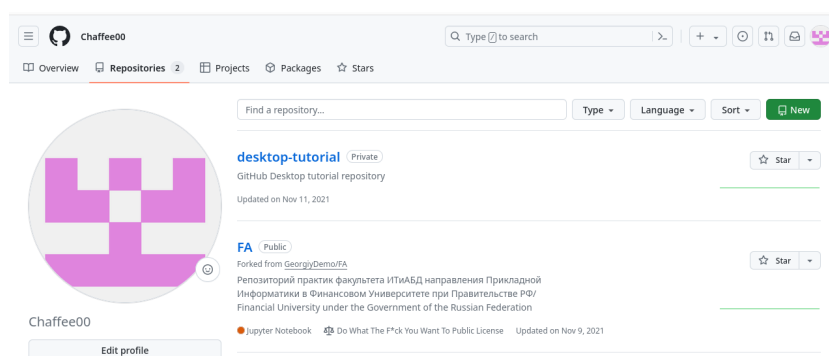


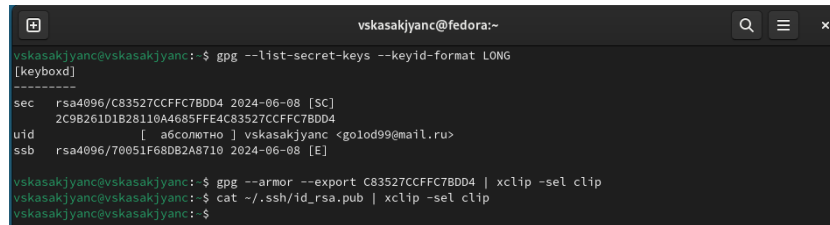
Рис. 4.5: Учетная запись GitHub

Выводим список ключей и копируем опечаток приватного ключа. Чтобы вывести список используем команду `gpg --list-secret-keys --keyid-format LONG`

Опечаток ключа находится в строке:

**sec Алгоритм/Отпечаток\_ключа Дата\_создания [Флаги] [Годен\_до]**

Копируем его командой `gpg --armor --export <Отпечаток_ключа> | xclip -sel clip` (рис. 4.6).



```
vskasakjanc@fedora:~  
vskasakjanc@vskasakjanc:~$ gpg --list-secret-keys --keyid-format LONG  
[keyboxd]  
-----  
sec  rsa4096/C83527CCFFC7BDD4 2024-06-08 [SC]  
      2C9B261D1B28110A4685FFE4C83527CCFFC7BDD4  
uid   [  _абсолютно ] vskasakjanc <golod99@mail.ru>  
ssb   rsa4096/78051F680B2A8710 2024-06-08 [E]  
  
vskasakjanc@vskasakjanc:~$ gpg --armor --export C83527CCFFC7BDD4 | xclip -sel clip  
vskasakjanc@vskasakjanc:~$ cat ~/.ssh/id_rsa.pub | xclip -sel clip  
vskasakjanc@vskasakjanc:~$
```

Рис. 4.6: Копирование pgr ключа

После скопированный ключ добавляем на GitHub (**settings** → **SSH and GPG keys** → **New GPG key**). Подобным образом добавляем и ssh ключ (рис. 4.7).

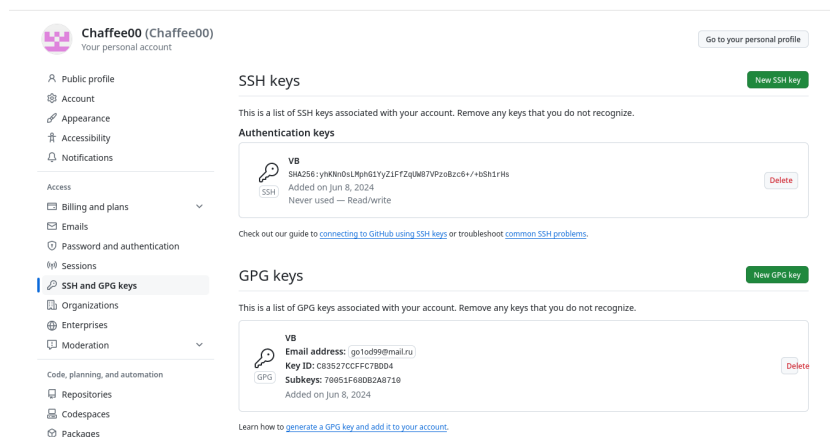
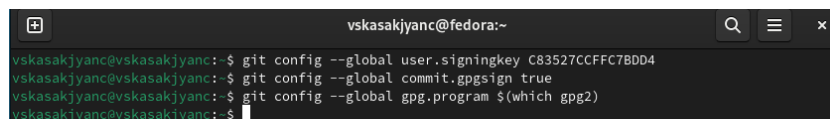


Рис. 4.7: Добавление ключей

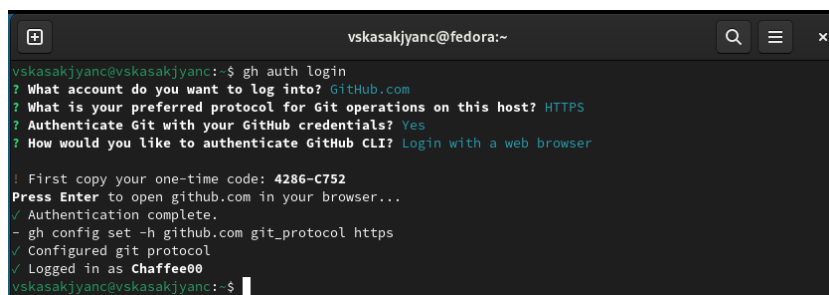
Настроим автоматические подписи коммитов git с помощью команд: `git config --global user.signingkey <Отпечаток_ключа>`, `git config --global commit.gpgsign true`, `git config --global gpg.program $(which gpg2)` (рис. 4.8).



```
vskasakjanc@fedora:~  
vskasakjanc@vskasakjanc:~$ git config --global user.signingkey C83527CCFFC7BDD4  
vskasakjanc@vskasakjanc:~$ git config --global commit.gpgsign true  
vskasakjanc@vskasakjanc:~$ git config --global gpg.program $(which gpg2)  
vskasakjanc@vskasakjanc:~$
```

Рис. 4.8: Настройка коммитов

Дальше стоит настроить gh. Для этого вводим команду `gh auth login`. Авторизируемся через браузер, вводим код из терминала и все готово (рис. 4.9), (рис. 4.10).



```
vskasakjyanc@fedora:~  
vskasakjyanc@vskasakjyanc:~$ gh auth login  
? What account do you want to log into? GitHub.com  
? What is your preferred protocol for Git operations on this host? HTTPS  
? Authenticate Git with your GitHub credentials? Yes  
? How would you like to authenticate GitHub CLI? Login with a web browser  
! First copy your one-time code: 4286-C752  
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 Chaffee00  
vskasakjyanc@vskasakjyanc:~$
```

Рис. 4.9: Авторизация

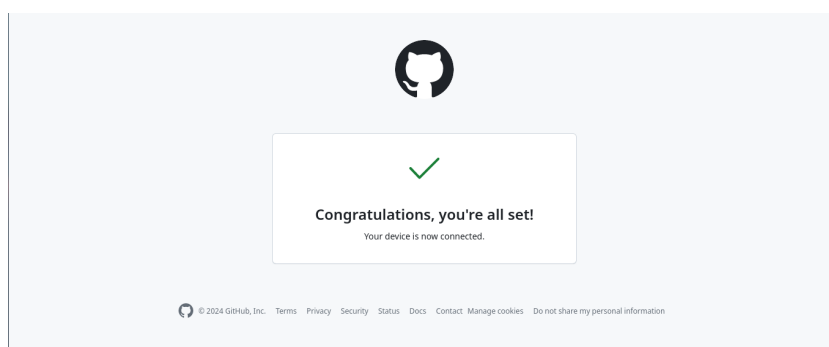


Рис. 4.10: Авторизация

Создадим репозиторий на GitHub. Для 2023–2024 учебного года и предмета «Операционные системы» (код предмета `os-intro`) создание репозитория примет следующий вид:

```
mkdir -p ~/work/study/2023-2024/"Операционные системы"  
cd ~/work/study/2023-2024/"Операционные системы"  
gh repo create study_2023-2024_os-intro --template=yamadharma/course-  
directory-student-template --public  
git clone --recursive git@github.com:<owner>/study_2023-2024_os-  
intro.git os-intro (рис. 4.12).
```

```
vskasakjyanc@fedora:~/work/study/2023-2024/Операционные системы
vskasakjyanc@vskasakjyanc:~$ mkdir -p ~/work/study/2023-2024/Операционные системы
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы$ cd work/study/2023-2024/Операционные системы/
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы$ gh repo create study_2023-2024_os-intro
--template=yamadharma/course-directory-student-template --public
✓ Created repository Chaffee00/study_2023-2024_os-intro on GitHub
https://github.com/Chaffee00/study_2023-2024_os-intro
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы$ git clone --recursive git@github.com:Chaffee00/study_2023-2024_os-intro.git os-intro
Клонирование в «os-intro»...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 32 (delta 1), reused 18 (delta 0), pack-reused 0
Получение объектов: 100% (32/32), 18.60 КиБ | 9.30 МБ/с, готово.
Определение изменений: 100% (1/1), 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/vskasakjyanc/work/study/2023-2024/Операционные системы/os-intro/template/presentation»...
remote: Enumerating objects: 95, done.
remote: Counting objects: 100% (95/95), done.
remote: Compressing objects: 100% (67/67), done.
remote: Total 95 (delta 34), reused 87 (delta 26), pack-reused 0
Получение объектов: 100% (95/95), 96.99 КиБ | 1.21 МБ/с, готово.
Определение изменений: 100% (34/34), готово.
Клонирование в «/home/vskasakjyanc/work/study/2023-2024/Операционные системы/os-intro/template/report»...
remote: Enumerating objects: 126, done.
remote: Counting objects: 100% (126/126), done.
remote: Compressing objects: 100% (87/87), done.
remote: Total 126 (delta 52), reused 108 (delta 34), pack-reused 0
Получение объектов: 100% (126/126), 335.80 КиБ | 2.01 МБ/с, готово.
Определение изменений: 100% (52/52), готово.
Submodule path 'template/presentation': checked out '40a1761813e197d00e8443ff1ca72c60a304f24c'
Submodule path 'template/report': checked out '7c31ab8e5dfa8cdb2d67cae8a19ef8028ced88e'
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы$
```

Рис. 4.11: Создание репозитория

Настроим каталог курса. Для этого сначала перейдем в сам каталог, который клонировали до этого, удалим лишние файлы, также создадим еще необходимые каталоги, которые после отправим на сервер (рис. 4.12), (рис. 4.13).

```
vskasakjyanc@fedora:~/work/study/2023-2024/Операционные системы/os-intro$ cd os-intro/
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы/os-intro$ rm package.json
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы/os-intro$ ls
CHANGELOG.md  config  COURSE  LICENSE  Makefile  README.en.md  README.git-flow.md  README.md  template
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы/os-intro$ echo os-intro > COURSE
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы/os-intro$ make

Usage:
  make <target>

Targets:
  list              List of courses
  prepare           Generate directories structure
  submodule         Update submules

vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы/os-intro$ make prepare
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы/os-intro$ make submodule
git submodule update --init --recursive
git submodule foreach 'git fetch origin; git checkout $(git rev-parse --abbrev-ref HEAD); git reset --hard orig
in$(git rev-parse --abbrev-ref HEAD); git submodule update --recursive; git clean -dfx'
Entering 'template/presentation'
Указатель HEAD сейчас на коммите 40a1761 Merge branch 'release/1.0.3'
Entering 'template/report'
Указатель HEAD сейчас на коммите 7c31ab8 Merge branch 'release/1.0.4'
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы/os-intro$ git add .
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы/os-intro$ git commit -am 'feat(main): mak
e course structure'
[master 4eba0c6] feat(main): make course structure
361 files changed, 98413 insertions(+), 14 deletions(-)
create mode 100644 labs/README.md
create mode 100644 labs/README.ru.md
create mode 100644 labs/lab01/presentation/Makefile
create mode 100644 labs/lab01/presentation/image/kulyabov.jpg
create mode 100644 labs/lab01/presentation/presentation.md
create mode 100644 labs/lab01/report/Makefile
create mode 100644 labs/lab01/report/bib/cite.bib
create mode 100644 labs/lab01/report/image/placeimg_800_600_tech.jpg
create mode 100644 labs/lab01/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 100755 labs/lab01/report/pandoc/filters/pandoc_eqnos.py
create mode 100755 labs/lab01/report/pandoc/filters/pandoc_fignos.py
create mode 100755 labs/lab01/report/pandoc/filters/pandoc_secnos.py
create mode 100755 labs/lab01/report/pandoc/filters/pandoc_tablenos.py
create mode 100644 labs/lab01/report/pandoc/filters/pandocxnos/__init__.py
create mode 100644 labs/lab01/report/pandoc/filters/pandocxnos/core.py
create mode 100644 labs/lab01/report/pandoc/filters/pandocxnos/main.py
create mode 100644 labs/lab01/report/pandoc/filters/pandocxnos/pandocattributes.py
create mode 100644 labs/lab01/report/report.md
```

Рис. 4.12: Настройка каталога курса

```
vskasakjyanc@fedora:~/work/study/2023-2024/Операционные системы/os-intro$ git add .
create mode 100644 project-personal/stage4/report/pandoc/filters/pandocxnos/main.py
create mode 100644 project-personal/stage4/report/pandoc/filters/pandocxnos/pandocattributes.py
create mode 100644 project-personal/stage4/report/report.md
create mode 100644 project-personal/stage5/presentation/Makefile
create mode 100644 project-personal/stage5/presentation/image/kulyabov.jpg
create mode 100644 project-personal/stage5/presentation/presentation.md
create mode 100644 project-personal/stage5/report/Makefile
create mode 100644 project-personal/stage5/report/bib/cite.bib
create mode 100644 project-personal/stage5/report/image/placeimg_800_600_tech.jpg
create mode 100644 project-personal/stage5/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 100755 project-personal/stage5/report/pandoc/filters/pandoc_eqnos.py
create mode 100755 project-personal/stage5/report/pandoc/filters/pandoc_fignos.py
create mode 100755 project-personal/stage5/report/pandoc/filters/pandoc_secnos.py
create mode 100755 project-personal/stage5/report/pandoc/filters/pandoc_tablenos.py
create mode 100644 project-personal/stage5/report/pandoc/filters/pandocxnos/__init__.py
create mode 100644 project-personal/stage5/report/pandoc/filters/pandocxnos/core.py
create mode 100644 project-personal/stage5/report/pandoc/filters/pandocxnos/main.py
create mode 100644 project-personal/stage5/report/pandoc/filters/pandocxnos/pandocattributes.py
create mode 100644 project-personal/stage5/report/report.md
create mode 100644 project-personal/stage6/presentation/Makefile
create mode 100644 project-personal/stage6/presentation/image/kulyabov.jpg
create mode 100644 project-personal/stage6/presentation/presentation.md
create mode 100644 project-personal/stage6/report/Makefile
create mode 100644 project-personal/stage6/report/bib/cite.bib
create mode 100644 project-personal/stage6/report/image/placeimg_800_600_tech.jpg
create mode 100644 project-personal/stage6/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 100755 project-personal/stage6/report/pandoc/filters/pandoc_eqnos.py
create mode 100755 project-personal/stage6/report/pandoc/filters/pandoc_fignos.py
create mode 100755 project-personal/stage6/report/pandoc/filters/pandoc_secnos.py
create mode 100755 project-personal/stage6/report/pandoc/filters/pandoc_tablenos.py
create mode 100644 project-personal/stage6/report/pandoc/filters/pandocxnos/__init__.py
create mode 100644 project-personal/stage6/report/pandoc/filters/pandocxnos/core.py
create mode 100644 project-personal/stage6/report/pandoc/filters/pandocxnos/main.py
create mode 100644 project-personal/stage6/report/pandoc/filters/pandocxnos/pandocattributes.py
create mode 100644 project-personal/stage6/report/report.md
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы/os-intro$ git push
Перечисление объектов: 40, готово.
Подсчет объектов: 100% (40/40), готово.
При сжатии изменений используется до 2 потоков
Сжатие объектов: 100% (30/30), готово.
Запись объектов: 100% (38/38), 342.11 КиБ | 2.87 Миб/с, готово.
Total 38 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To github.com:Chaffee00/study_2023-2024_os-intro.git
af5824c..4eba0c6 master -> master
vskasakjyanc@vskasakjyanc:~/work/study/2023-2024/Операционные системы/os-intro$
```

Рис. 4.13: Отправка на сервер

## 5 Контрольные вопросы

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

Система контроля версий (VCS) — это инструмент, использование которого позволяет отслеживать изменения в файловой системе, фиксировать историю изменений, а также возвращаться к предыдущим версиям файлов. Они предназначены для управления изменениями в проектах программного обеспечения и других файлов, позволяя команде разработчиков совместно работать над кодом, отслеживать изменения, управлять конфликтами и версиями, а также восстанавливать предыдущие состояния проекта.

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

**Хранилище (репозиторий)** — это централизованное место, где хранятся файлы и история изменений проекта. Оно содержит все версии файлов, метаданные и историю коммитов.

**Commit (фиксация)** — это действие по сохранению изменений в системе контроля версий. При коммите разработчик предоставляет описание внесенных изменений, и эти изменения фиксируются в репозитории.

**История (history)** — это список всех коммитов и изменений, связанных с проектом. История содержит информацию о том, кто, когда и какие изменения вносил, и позволяет отслеживать всю историю проекта.



**Рабочая копия (working copy)** — это локальная копия файлов из репозитория, с которой работает разработчик. Рабочая копия содержит текущую версию проекта, и разработчик вносит изменения в нее перед их фиксацией (коммитом) в репозиторий.

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

**Централизованная система контроля версий (Centralized Version Control System, CVS)** предполагает, что существует единый центральный репозиторий, в котором хранится вся история проекта. Разработчики работают с рабочими копиями файлов, которые забирают из центрального репозитория, вносят изменения и отправляют их обратно. Примеры централизованных VCS: CVS, Subversion (SVN), Perforce.

**Децентрализованные системы контроля версий (Distributed Version Control System, DVCS)** отличаются тем, что каждый разработчик имеет свою локальную копию репозитория, содержащую всю историю проекта. Это позволяет работать независимо от доступности центрального сервера и облегчает совместную работу над проектом. Примеры децентрализованных VCS: Git, Mercurial, Bazaar.

Основные отличия между централизованными и децентрализованными системами контроля версий заключаются в том, как управляется и хранится история версий проекта, а также в способе совместной работы разработчиков.

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

При единоличной работе с хранилищем VCS основными действиями будут:

- **Инициализация репозитория:** создание нового проекта или клонирование существующего репозитория из удаленного источника (например, GitHub).

- **Добавление файлов:** добавление новых файлов в репозиторий или изменение уже существующих файлов.
- **Фиксация изменений:** коммит изменений в репозиторий с указанием описания изменений.
- **Просмотр истории изменений:** просмотр и анализ всех предыдущих коммитов, внесенных в репозиторий.
- **Ветвление:** создание отдельных веток для разработки новых функций или исправлений багов.
- **Объединение изменений:** слияние веток и консолидация изменений в основной ветке разработки.
- **Удаление файлов:** удаление ненужных файлов из репозитория.
- **Удаленная работа:** отправка изменений на удаленный сервер и получение изменений из удаленного репозитория.

Все эти действия помогают эффективно управлять версиями кода и отслеживать изменения в проекте, даже при работе в одиночку.

## 5. Опишите порядок работы с общим хранилищем VCS.

- **Создание репозитория:** сначала необходимо создать репозиторий на сервере или в облаке, где будет храниться общее хранилище файлов.
- **Клонирование репозитория:** разработчики должны клонировать репозиторий себе на локальную машину, чтобы иметь доступ к файлам и иметь возможность вносить изменения.
- **Работа с файлами:** разработчики могут вносить изменения в файлы на локальной машине, создавать новые файлы, удалять или редактировать существующие.

- **Подготовка к коммиту:** перед сохранением изменений в репозиторий, необходимо подготовить их к коммиту, добавив их в “индекс” при помощи команды `git add`.
- **Коммит изменений:** после подготовки изменений, разработчики должны сделать коммит, сохраняя все внесенные изменения в историю репозитория при помощи команды `git commit`.
- **Пуш изменений:** после коммита, изменения могут быть отправлены в общее хранилище с помощью команды `git push`, что позволит другим разработчикам видеть и получать эти изменения.
- **Обновление локального репозитория:** разработчики могут получить последние изменения из общего хранилища с помощью команды `git pull`, чтобы обновить свою локальную версию репозитория.

Таким образом, порядок работы с общим хранилищем VCS заключается в клонировании, внесении изменений, коммите и отправке изменений в общее хранилище, а также в получении и обновлении локальной версии репозитория.

## 6. Каковы основные задачи, решаемые инструментальным средством `git`?

- Управление версиями файлов и их изменениями
- Совместная разработка проектов
- Отслеживание изменений и истории проекта
- Управление конфликтами при слиянии изменений
- Резервное копирование и восстановление данных

## 7. Назовите и дайте краткую характеристику командам `git`.

**git init:** инициализация нового репозитория

**git add:** добавление изменений в индекс

**git commit:** сохранение изменений в репозитории

**git push:** отправка изменений в удаленный репозиторий

**git pull:** получение изменений из удаленного репозитория

**git branch:** создание, удаление и просмотр веток

**git merge:** объединение изменений из другой ветки

**git checkout:** переключение между ветками

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

**Локальный репозиторий:** создание нового проекта с помощью `git init`, добавление новых файлов с помощью `git add`, сохранение изменений в репозитории с помощью `git commit`.

**Удаленный репозиторий:** отправка изменений из локального репозитория на удаленный с помощью `git push`, получение изменений из удаленного репозитория с помощью `git pull`.

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

Ветви (branches) в `git` используются для разработки новых функций, изоляции изменений, параллельной разработки, исправления ошибок и управления версиями проектов.

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

Для игнорирования некоторых файлов при `commit` в `git` используется файл `.gitignore`, в котором указываются шаблоны файлов или директорий, которые не должны попадать в репозиторий. Например, можно игнорировать временные файлы, файлы с настройками IDE, файлы с конфиденциальной информацией и т.д.

## 6 Выводы

В данной лабораторной работе мы изучили идеологию и применение средств контроля версий, а также освоили умения по работе с git.

## Список литературы

1. О системе контроля версий [Электронный ресурс]. 2016. URL: <https://git-scm.com/book/ru/v2/Введение-О-системе-контроля-версий>.
2. Евгений Г. Системы контроля версий [Электронный ресурс]. 2016. URL: [https://glebradchenko.susu.ru/courses/bachelor/engineering/2016/SUSU\\_SE\\_2016\\_REP\\_3\\_VC](https://glebradchenko.susu.ru/courses/bachelor/engineering/2016/SUSU_SE_2016_REP_3_VC)
3. Системы контроля версий [Электронный ресурс]. 2016. URL: <http://uii.mpei.ru/study/cours>