

Отчёт по лабораторной работе №2

Операционные системы

Екатерина Павловна Канева

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Настройка GitHub	6
2.2	Базовая настройка git.	6
2.3	Создание SSH ключа.	8
2.4	Создание рабочего пространства и репозитория курса на основе шаблона.	9
2.5	Настройка каталога курса	10
2.6	Контрольные вопросы	13
3	Выводы	19

Список иллюстраций

2.1	Учётная запись на сайте GitHub	6
2.2	Ввод имени и адреса электронной почты.	7
2.3	Настройка utf-8 в выводе сообщений git.	7
2.4	Задаём имя начальной ветки.	7
2.5	Параметр autocrlf.	7
2.6	Параметр safecrlf.	7
2.7	Генерация SSH-ключа.	8
2.8	Копирование ключа.	8
2.9	Добавление нового SSH-ключа.	9
2.10	Выбор шаблона для репозитория.	9
2.11	Коммиты на github.	10
2.12	Генерация gpg ключа (часть 1).	11
2.13	Генерация gpg ключа (часть 2).	11
2.14	Экспорт gpg ключа.	12
2.15	Ввод gpg ключа на github.	12
2.16	Настройка автоматических подписей коммитов.	13
2.17	Флаг на неverified коммиты.	13
2.18	Вход в gh	13

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Настройка GitHub

Предварительно мною уже был создан аккаунт на сайте <https://github.com>, а также была заполнена основная информация (рис. 2.1):

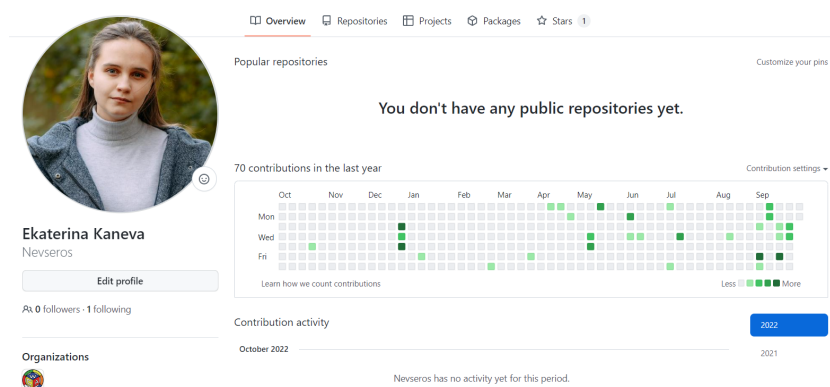


Рис. 2.1: Учётная запись на сайте GitHub

2.2 Базовая настройка git.

Сначала сделаем предварительную конфигурацию git. Откроем терминал и введём следующие команды, указав свои имя и email (рис. 2.2):

```
git config --global user.name "Ekaterina Kaneva"
git config --global user.email "nkanevan@gmail.com"
```

```
[epkaneva@fedora ~]$ git config --global user.name "Ekaterina Kaneva"
[epkaneva@fedora ~]$ git config --global user.email "nkanevan@gmail.com"
```

Рис. 2.2: Ввод имени и адреса электронной почты.

Настроим utf-8 в выводе сообщений git (рис. 2.3):

```
git config --global core.quotepath false
```

```
[epkaneva@fedora ~]$ git config --global core.quotepath false
```

Рис. 2.3: Настройка utf-8 в выводе сообщений git.

Зададим имя начальной ветки – master (рис. 2.4):

```
git config --global init.defaultBranch master
```

```
[epkaneva@fedora ~]$ git config --global init.defaultBranch master
```

Рис. 2.4: Задаём имя начальной ветки.

Параметр autocrlf (рис. 2.5) и safecrlf (рис. 2.6):

```
git config --global core.autocrlf input
```

```
git config --global core.safecrlf warn
```

```
[epkaneva@fedora ~]$ git config --global core.autocrlf input
```

Рис. 2.5: Параметр autocrlf.

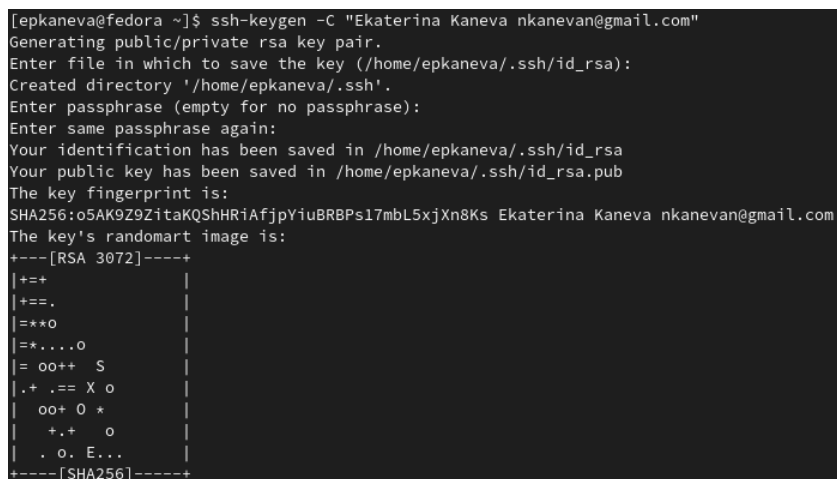
```
[epkaneva@fedora ~]$ git config --global core.safecrlf warn
```

Рис. 2.6: Параметр safecrlf.

2.3 Создание SSH ключа.

Для последующей идентификации на сервере репозитория сгенерируем пару ключей – приватный и открытый (рис. 2.7):

```
ssh-keygen -C "Ekaterina Kaneva nkanevan@gmail.com"
```

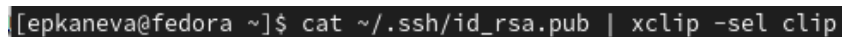
A terminal window showing the execution of the ssh-keygen command. The output includes prompts for a passphrase and confirmation of the key's location. It also displays the key's fingerprint and a randomart image.

```
[epkaneva@fedora ~]$ ssh-keygen -C "Ekaterina Kaneva nkanevan@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/epkaneva/.ssh/id_rsa):
Created directory '/home/epkaneva/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/epkaneva/.ssh/id_rsa
Your public key has been saved in /home/epkaneva/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:o5AK9Z9ZitakQShHRiAfjpYiuBRBPsl7mbL5xjXn8Ks Ekaterina Kaneva nkanevan@gmail.com
The key's randomart image is:
+---[RSA 3072]-----+
|+==+|
|+==.|
|==*o|
|==...o|
|== oo++ S|
|.+.== X o|
| oo+ 0 *|
| .+. o|
| . o. E...|
+---[SHA256]-----+
```

Рис. 2.7: Генерация SSH-ключа.

Далее, чтобы добавить новый сгенерированный ключ, авторизуемся на сайте [github.org](https://github.com) и введём новый ключ в настройках. Чтобы скопировать ключ, в консоль введём следующую команду, а затем скопируем ключ (рис. 2.8):

```
cat ~/.ssh/id_rsa.pub
```

A terminal snippet showing the command to copy the public key to the clipboard using xclip.

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

Рис. 2.8: Копирование ключа.

Теперь добавим ключ (рис. 2.9):

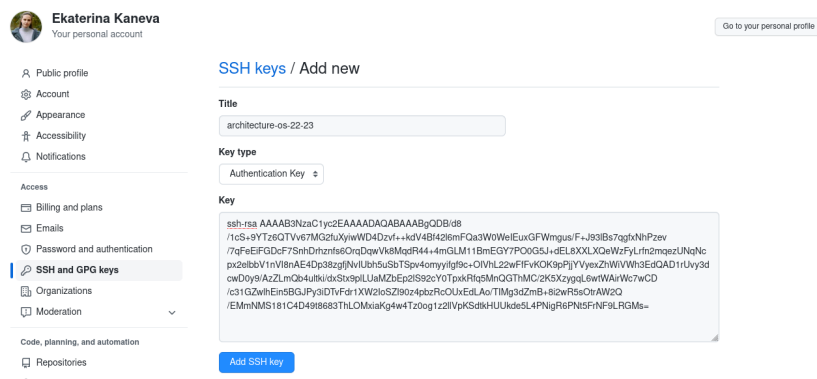


Рис. 2.9: Добавление нового SSH-ключа.

2.4 Создание рабочего пространства и репозитория курса на основе шаблона.

При выполнении первой лабораторной работы было создано рабочее пространство для предмета “Операционные системы” с помощью команды:

```
mkdir -p ~/work/study/2022-2023/«Операционные системы»
```

Используем указанный в тексте лабораторной работы шаблон для собственного репозитория (рис. 2.10):

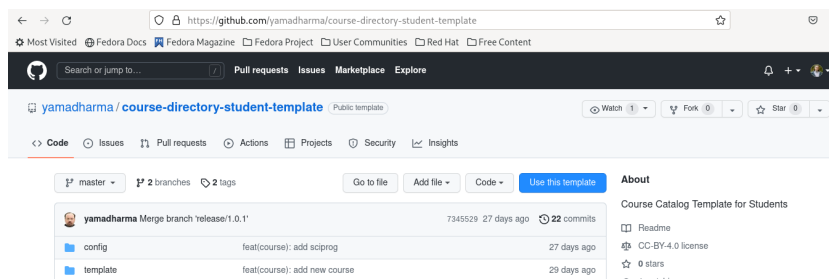


Рис. 2.10: Выбор шаблона для репозитория.

В открывшемся окне зададим имя репозитория `study_2022-2023_os-intro` и создадим репозиторий.

Откроем терминал, перейдём в каталог курса и клонируем репозиторий с помощью следующих команд:

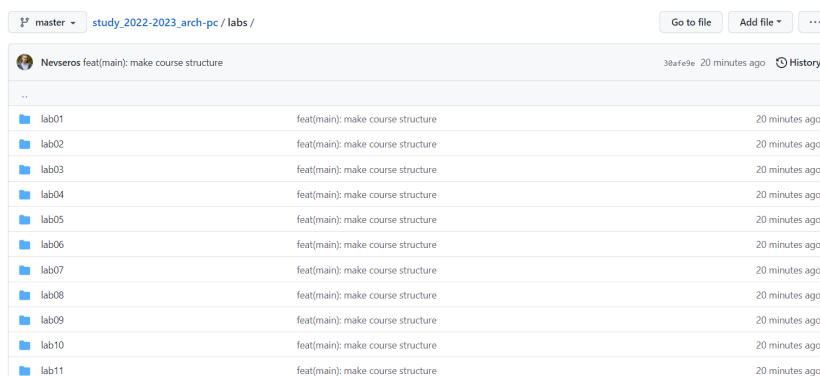
```
cd ~/work/study/2022-2023/"Операционные системы"
git clone --recursive git@github.com:Nevseros/study_2022-2023_os-
intro.git os-intro
```

2.5 Настройка каталога курса

Удалим лишние файлы и создадим необходимые каталоги. Отправим файлы на сервер (всё это делалось перед выполнением первой лабораторной, поэтому скриншотов ввода команд в терминале нет:

```
rm package.json
echo os-intro > COURSE
make
git add .
git commit -am 'feat(main): make course structure'
git push
```

Теперь проверим, что файлы действительно отправились на сервер (рис. 2.11):



The screenshot shows a GitHub commit page for the repository 'study_2022-2023_arch-pc / labs /'. The commit message is 'feat(main): make course structure' and it was made 20 minutes ago. The commit includes 11 files, all named 'lab01' through 'lab11', each with a commit hash and a timestamp of '20 minutes ago'.

File	Commit Hash	Time
lab01	feat(main): make course structure	20 minutes ago
lab02	feat(main): make course structure	20 minutes ago
lab03	feat(main): make course structure	20 minutes ago
lab04	feat(main): make course structure	20 minutes ago
lab05	feat(main): make course structure	20 minutes ago
lab06	feat(main): make course structure	20 minutes ago
lab07	feat(main): make course structure	20 minutes ago
lab08	feat(main): make course structure	20 minutes ago
lab09	feat(main): make course structure	20 minutes ago
lab10	feat(main): make course structure	20 minutes ago
lab11	feat(main): make course structure	20 minutes ago

Рис. 2.11: Коммиты на github.

Далее началось выполнение новой части лабораторной работы - был сгенерирован grpg ключ. Для этого была введена следующая команда (рис. 2.12 и 2.13):

```
[epkaneva@epkaneva lab02]$ gpg --full-generate-key
gpg (GnuPG) 2.3.7; 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.

Please select what kind of key you want:
  (1) RSA and RSA
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
  (9) ECC (sign and encrypt) *default*
 (10) ECC (sign only)
 (14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096
Requested keysize is 4096 bits
Please specify how long the key should be valid.
    0 = key does not expire
    <n> = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Ekaterina Kaneva
Email address: nkanevan@gmail.com
Comment:
You selected this USER-ID:
    "Ekaterina Kaneva <nkanevan@gmail.com>"
```

Рис. 2.12: Генерация gpg ключа (часть 1).

```
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /home/epkaneva/.gnupg/trustdb.gpg: trustdb created
gpg: directory '/home/epkaneva/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/epkaneva/.gnupg/openpgp-revocs.d/1FAA199B147D53B8E403936E6B35594658EB89A5.rev'
public and secret key created and signed.

pub   rsa4096 2023-02-18 [SC]
       1FAA199B147D53B8E403936E6B35594658EB89A5
uid    Ekaterina Kaneva <nkanevan@gmail.com>
sub    rsa4096 2023-02-18 [E]
```

Рис. 2.13: Генерация gpg ключа (часть 2).

Далее для экспорта ключа была введена команда:

```
gpg --armor --export 6B35594658EB89A5
```

Где 6B35594658EB89A5 – отпечаток ключа.

```
[epkaneva@epkaneva lab02]$ gpg --armor --export 6B35594658EB89A5
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGPwylIBEAD03Va5arpvUMpV5iFvEBcHluooUkaP2iJ2HL6lfve3R3iCwxEd
b0bd80NkCT/sly/wT28ChPpUs88BEMF3ET0fIzItX08eeY/A2cg062gLuEtT+CZx
Gb0gzJhKwFpQUwiHlomLH2PbK0gEJfAh2/lyWQZ7HzKDTyamiWl4vqVgRCreOUMW
VWj fPvc29PVQ3pal2mXj0kNJBbut6eit5W2EZYiv7QxcexSkT/snfby6H8IR1W7x
HJiC6vvw/PcwpeBpeHWPZ6QfqLudhx16cIx/kMhWjYVYcTHBBPR2r47LZZAQum9K
dLXUabKC2GY4pBct0/BGsBHCX5CebyhqlwnNwmfZ2Mvw05qaLD6js+VktXAsVBga
LkrkyeBJLIT8+A00kdRpVfdhJnf9cmYBnpFn3wd3WhTShLK5kIZFRawbNqBwhv6W
1IdXJQ/bH0fDeLCAzqWF9386tyNfHem0yGUB54D2iPyfbEU+68hcRYvG0Enp4uV2
qbQAN6WrTunCq008z0ddzogo0z/DjPl2I/64k4QXq7yxyZk50/hRzvFcEezW1hZo
BCFXBUFWGvY5MufdD9oD6lFMcY8PJV20mLvRQ80vHtL27/EjG3eBKE8VdFqMVjZy
HbVnazVeObjEDWPTIqNEU8XRS0reSDmnfk9VZr3w8rBsK6Q2mIR0FFZU6QARAQAB
tCVFa2F0ZXJpbmEgS2FuZXZhIDxua2FuZXZhbkBnbWVpbC5jb20+iQJRBBMBCAA7
FiEEH6oZmxR9U7jka5NuazVZRljriaUFAmPwylICGwMFCwkIBwICIGIGFQoJCAcC
BBYCAwECHgcCF4AACgkQazVZRljriaVcSRAAhLHdYwbZgY+IPyIzcZFRPGwEK98F
r/OiBKsFW7Gaei/bIfn26F0MY73W0teRDbN6K/lR0hzU/O3CWlV0f9bCgoTx05Wf
QreXZSV8JKu6gzS6S4cj4ijU7f8IXACD0qHd1HeIUia2BfjnPGA1Jmnw0bYtvztK
uj1r7z50Ihm8Y7x1wp4DNw1hU+Bh1ETeIzLUeRbtY0h662nB/3Aw0J7JpdR/mW+M
eewlmrepwtUKN4sRKwscFU9ET2wXVy4w9GoxYQz09z5XWPWPz1w9w/zD3H3jRH2j
hYq0z2IZVxzq4y6EpS38DAL03Q5yw+lJ4WuC10ZN4SJS0zT4Ek9z90ZkyVKRMPH
riUHHJevLLQLmCHIYIkMlqG8PlzMOts2EWwbuYx0tSTMGE98XI tFJ2c i oVR5QXK+
```

Рис. 2.14: Экспорт gpg ключа.

Позже этот ключ был введён на сайте github (рис. 2.15):

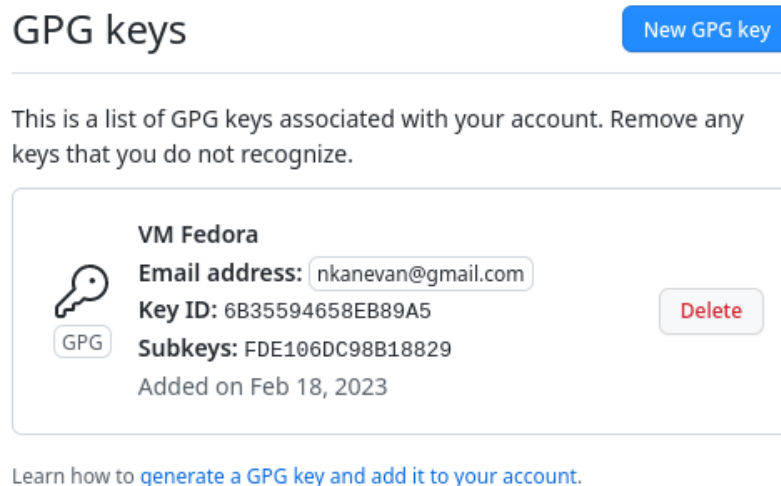


Рис. 2.15: Ввод gpg ключа на github.

Были настроены автоматические подписи коммитов (рис. 2.16):

```
[epkaneva@epkaneva lab02]$ git config --global user.signinkey 6B35594658EB89A5
[epkaneva@epkaneva lab02]$ git config --global commit.gpgsign true
[epkaneva@epkaneva lab02]$ git config --global gpg.program $(which gpg2)
```

Рис. 2.16: Настройка автоматических подписей коммитов.

Неверифицированные коммиты теперь выделяются (рис. 2.17):

Vigilant mode

☒ Flag unsigned commits as unverified

This will include any commit attributed to your account but not signed with your GPG or S/MIME key.

Note that this will include your existing unsigned commits.

Рис. 2.17: Флаг на неверифицированные коммиты.

Также был выполнен вход в gh (рис. 2.18):

```
[epkaneva@epkaneva lab02]$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? 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: E177-D543
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 Nevseros
```

Рис. 2.18: Вход в gh

2.6 Контрольные вопросы

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

Система контроля версий – это 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, в нём хранятся все документы, история их изменений и другая служебная информация.

commit – добавленные изменения.

История хранит все изменения в проекте и позволяет обратиться к нужным версиям документов.

Рабочая копия – текущее состояние файлов проекта.

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

Централизованные системы контроля версий предполагают наличие централизованного хранилища, а пользователи сами копируют все необходимые файлы к себе, добавляют изменения и отправляют их на централизованный репозиторий. Например, CVS, TFS.

Децентрализованные системы предполагают наличие у каждого пользователя своего репозитория (а может и не одного), изменения можно получить из любого из таких репозиториях. Например, Git, Bazaar.

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

Создаём репозиторий, по мере изменения проекта отправляем изменения (коммиты) на сервер (удалённый репозиторий).

5. Опишите порядок работы с общим хранилищем VCS.

Перед тем, как начинать изменять проект, надо получить актуальную версию. Затем можно вносить изменения, отправлять их на удалённый репозиторий.

6. Каковы основные задачи, решаемые инструментальным средством git?

Хранение файлов проекта и истории их изменения, а также удобство командной работы над проектами.

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 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
```

```
git switch -c
```

- переключение на некоторую ветку:

```
git checkout имя_ветки
```

```
git switch
```

- отправка изменений конкретной ветки в центральный репозиторий:

```
git push origin имя_ветки
```

- слияние ветки с текущим деревом:

```
git merge 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
```

- Переключение на старые коммиты:

- новым коммитом (можно отменить):

```
git revert имя_коммита
```

- без нового коммита

```
git reset --hard имя_коммита
```

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

Например, в каком-либо проекте внесли изменения в файлы, теперь нужно их отправить на сервер. Предварительно, перед внесением изменений, притянем изменения с удалённого репозитория, потом создадим новую ветку под нашу (допустим) feature, чтобы потом открыть pull request на github (или merge request, если gitlab):

```
git pull
```

```
git switch -c feature/redirect
```

```
git add .
```

```
git commit -m 'Add redirect route'
```

```
git push --set-upstream origin feature/redirect
```

После этого в удалённом репозитории появится возможность открыть pull/merge request. Там, по необходимости, нужно будет дождаться фидбэка от других контрибьюторов, решить конфликты веток, если таковые есть, а потом влить изменения.

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

Ветви – параллельные участки истории. Нужны, чтобы работать над разными заданиями одновременно, а также чтобы удобнее было работать нескольким людям над одним проектом, чтобы можно было открывать pull/merge request'ы и получать ревью кода.

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

Во время работы над проектом могут появляться ненужные файлы (например, какие-то временные файлы создаваемые редакторами или объектные файлы, создаваемые компиляторами). Их можно игнорировать с помощью .gitignore (специальный файл в репозитории). Можно также использовать команду `git reset HEAD имя_файла`, чтобы исключить его из текущего коммита (после `git add`).

3 Выводы

Изучили идеологию и применение системы контроля версий. Приобрели практические навыки работы с системой git.