
Front matter

title: "Отчёт по лабораторной работе №2"
subtitle: "Операционные системы"
author: "Волчкова Елизавета Дмитриевна"

Generic options

lang: ru-RU
toc-title: "Содержание"

Bibliography

bibliography: bib/cite.bib
csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl

Pdf output format

toc: true # Table of contents
toc-depth: 2
lof: true # List of figures
lot: true # List of tables
fontsize: 12pt
linestretch: 1.5
papersize: a4
documentclass: scrreprt

l18n polyglossia

polyglossia-lang:
name: russian

polyglossia-otherlangs:
name: english

l18n babel

babel-lang: russian
babel-otherlangs: english

Fonts

mainfont: IBM Plex Serif
romanfont: IBM Plex Serif
sansfont: IBM Plex Sans
monofont: IBM Plex Mono
mathfont: STIX Two Math
mainfontoptions: Ligatures=Common,Ligatures=TeX,Scale=0.94
romanfontoptions: Ligatures=Common,Ligatures=TeX,Scale=0.94
sansfontoptions: Ligatures=Common,Ligatures=TeX,Scale=MatchLowercase,Scale=0.94

monofontoptions: Scale=MatchLowercase,Scale=0.94,FakeStretch=0.9
mathfontoptions:

Biblatex

biblatex: true

biblio-style: "gost-numeric"

biblatexoptions:

- parenttracker=true
- backend=biber
- hyperref=auto
- language=auto
- autolang=other*
- citestyle=gost-numeric

Pandoc-crossref LaTeX customization

figureTitle: "Рис."

tableTitle: "Таблица"

listingTitle: "Листинг"

lofTitle: "Список иллюстраций"

lotTitle: "Список таблиц"

lolTitle: "Листинги"

Misc options

indent: true

header-includes:

- \usepackage{indentfirst}
- \usepackage{float} # keep figures where there are in the text
- \floatplacement{figure}{H} # keep figures where there are in the text

Цель работы

ЦЦель работы

Изучить идеологию и применение средств контроля версий.

Освоить умения по работе с git.

Теоретические сведения

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 дельта-компрессию — 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.

Примеры использования git

Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды `git` с различными опциями. Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией.

Основные команды git

Перечислим 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

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

Стандартные процедуры работы при наличии центрального репозитория

Работа пользователя со своей веткой начинается с проверки и получения изменений из центрального репозитория (при этом в локальное дерево до начала этой процедуры не должно было вноситься изменений):

```
git checkout master
```

```
git pull
```

```
git checkout -b имя_ветки
```

Затем можно вносить изменения в локальном дереве и/или ветке.

После завершения внесения какого-то изменения в файлы и/или каталоги проекта необходимо разместить их в центральном репозитории. Для этого необходимо проверить, какие файлы изменились к текущему моменту:

```
git status
```

При необходимости удаляем лишние файлы, которые не хотим отправлять в центральный репозиторий.

Затем полезно просмотреть текст изменений на предмет соответствия правилам ведения чистых коммитов:

```
git diff
```

Если какие-либо файлы не должны попасть в коммит, то помечаем только те файлы, изменения которых нужно сохранить. Для этого используем команды добавления и/или удаления с нужными опциями:

```
git add ...
```

```
git rm ...
```

Если нужно сохранить все изменения в текущем каталоге, то используем:

```
git add .
```

Затем сохраняем изменения, поясняя, что было сделано:

```
git commit -am "Some commit message"
```

Отправляем изменения в центральный репозиторий:

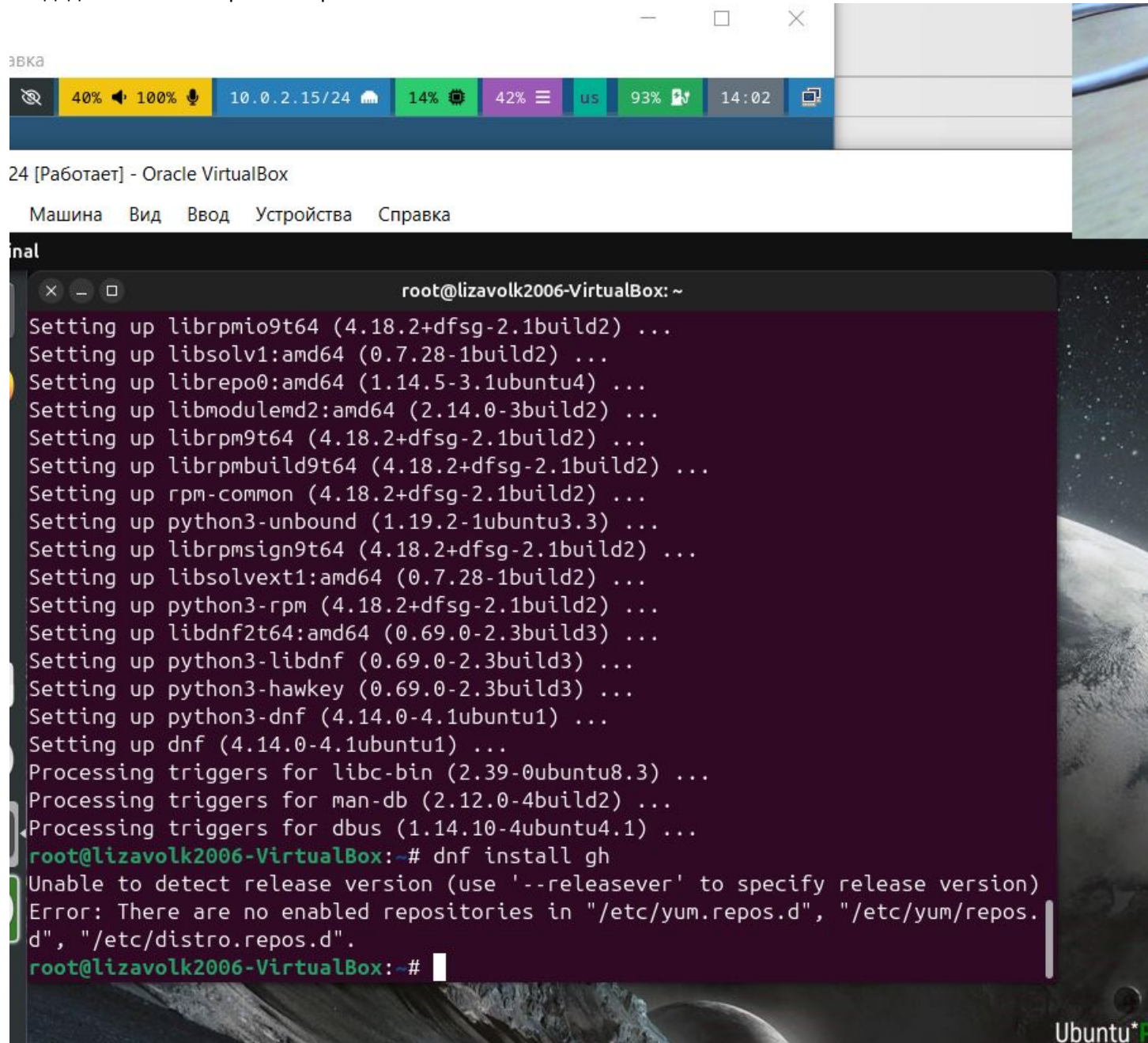
```
git push origin имя_ветки
```

или

```
git push
```

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"
```

Настроим utf-8 в выводе сообщений git:

```
1 foot 40% 100% 10.0.2.15/24 97% 47% us 92%
foot
Please type liveinst and press Enter to start the installer
[liveuser@localhost-live ~]$ dnf install gh
The requested operation requires superuser privileges. Please log in as a user with elevated rights, or use the "--assumeno" or "--downloadonly" options without modifying the system state.
[liveuser@localhost-live ~]$ sudo -i
[root@localhost-live ~]# dnf install gh
Updating and loading repositories:
  Fedora 41 openh264 (From Cisco) - x86_64 100% | 289.0 B/s
  Fedora 41 - x86_64 - Updates 100% | 10.4 MiB/s
  Fedora 41 - x86_64 100% | 8.0 MiB/s
Repositories loaded.
Package Arch Version Repository
Installing:
gh x86_64 2.65.0-1.fc41 updates
Installing dependencies:
git-core x86_64 2.48.1-1.fc41 updates

Transaction Summary:
Installing: 2 packages

Total size of inbound packages is 15 MiB. Need to download 15 MiB.
After this operation, 65 MiB extra will be used (install 65 MiB, remove 0 B).
Is this ok [y/N]: y
[1/2] git-core-0:2.48.1-1.fc41.x86_64 100% | 10.8 MiB/s
[2/2] gh-0:2.65.0-1.fc41.x86_64 100% | 12.2 MiB/s
-----
[2/2] Total 100% | 9.5 MiB/s
Running transaction
[1/4] Verify package files 100% | 31.0 B/s
[2/4] Prepare transaction 100% | 11.0 B/s
[3/4] Installing git-core-0:2.48.1-1.fc41.x86_64 100% | 85.3 MiB/s
[4/4] Installing gh-0:2.65.0-1.fc41.x86_64 100% [=====] | 97.6 MiB/s
>>> Running trigger-install scriptlet: glibc-common-0:2.40-3.fc41.x86_64warning: posix.fork(): .fork(), .exec(), .wait() and .redirect2null() are deprecated, use rpm.spawn() or rpm.execute() instead
warning: posix.wait(): .fork(), .exec(), .wait() and .redirect2null() are deprecated, use rpm.spawn() or rpm.execute() instead
[4/4] Installing gh-0:2.65.0-1.fc41.x86_64 100% | 5.6 MiB/s
Complete!
[root@localhost-live ~]#
```

git config --global quotePath false

Для инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial,

необходимо ввести в командной строке:

2024 [Работает] - Oracle VM VirtualBox

Файл Машина Вид Ввод Устройства Справка

```
Terminal
root@lizavolk2006-VirtualBox: ~
Enter same passphrase again:
Passphrases do not match. Try again.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Passphrases do not match. Try again.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ssh-keygen -t ed25519
Your public key has been saved in ssh-keygen -t ed25519.pub
The key fingerprint is:
SHA256:4TcAFYWHs7D1lciCmRAE1dfqLPBKXi09uoTf/IL4wY root@lizavolk2006-VirtualBox
The key's randomart image is:
+----[RSA 4096]-----+
|  .+*= + *.. . |
|  .B 0 = o |
|  o* B . |
|  ...oo . |
|  +Soo |
|  .E *.o. |
|  .o.=o+ |
|  oo=oo . |
|  ++*oo |
+-----[SHA256]-----+
root@lizavolk2006-VirtualBox: ~#
```

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

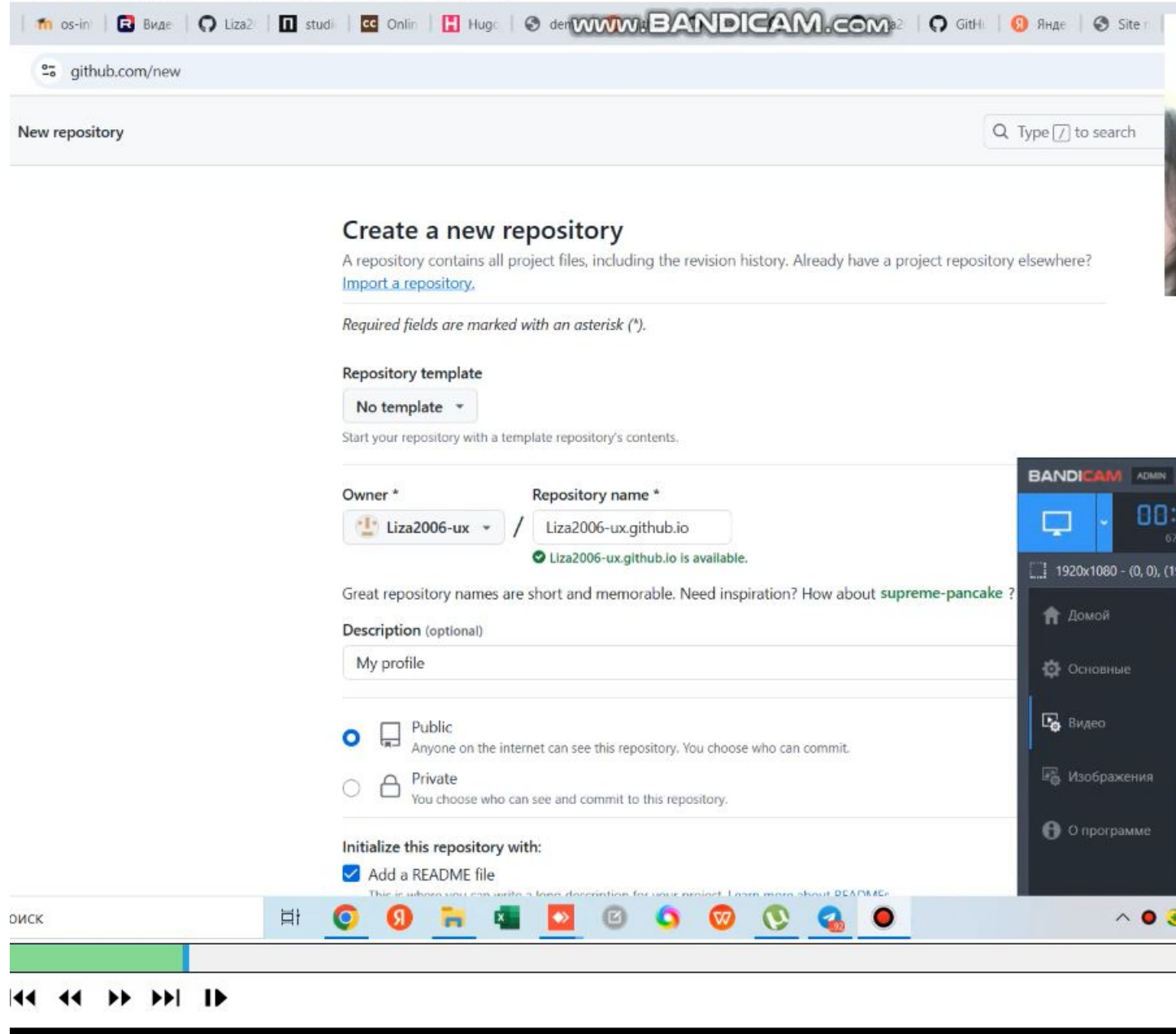
git add hello.txt

git commit -am 'Новый файл'

Воспользуемся командой status для просмотра изменений в рабочем каталоге, сделанных с

момента последней ревизии:

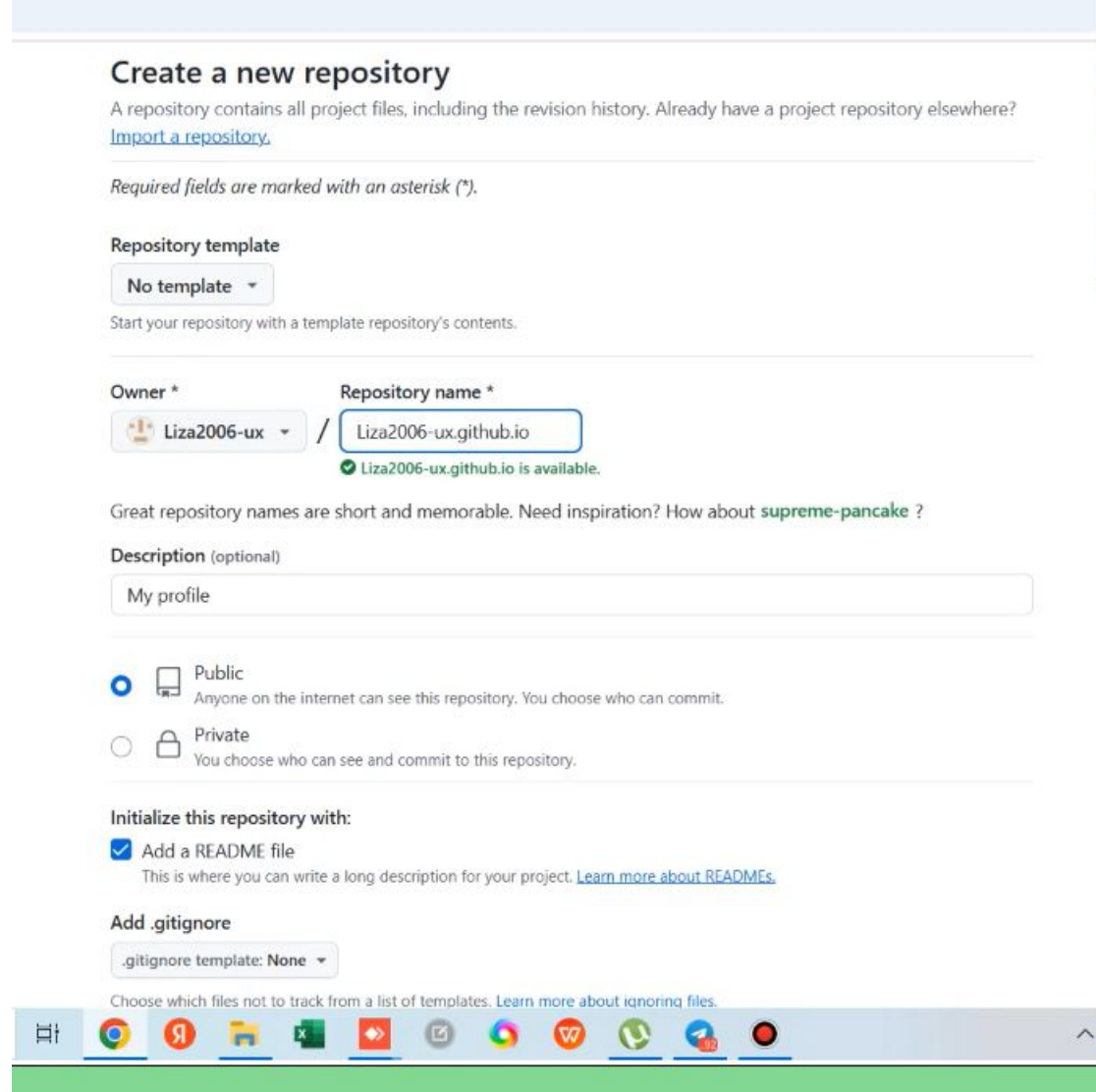
Воспроизведение Навигация Закладки Помощь



git status

Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл .gitignore с помощью сервисов. Для этого сначала нужно получить список имеющихся шаблонов:

curl -L -s https://www.gitignore.io/api/list
Затем скачать шаблон, например, для С и С++



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner * **Repository name ***

Liza2006-ux ▾ / Liza2006-ux.github.io

✓ Liza2006-ux.github.io is available.

Great repository names are short and memorable. Need inspiration? How about **supreme-pancake** ?

Description (optional)

My profile

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☒ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

```
curl -L -s https://www.gitignore.io/api/c >> .gitignore  
curl -L -s https://www.gitignore.io/api/c++ >> .gitignore
```

Работа с сервером репозиториев

Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый):

```
ssh-keygen -C "Лиза Волчкова @mail"
```

Ключи сохраняются в каталоге ~/.ssh/.

Существует несколько доступных серверов репозиториев с возможностью бесплатного размещения данных. Например, <https://github.com/>.

Для работы с ним сначала завела на сайте <https://github.com/> учётную запись. Загрузила сгенерённый нами ранее открытый ключ.

Для этого зашла на сайт <https://github.com/> под своей учётной записью и перешла в меню GitHub

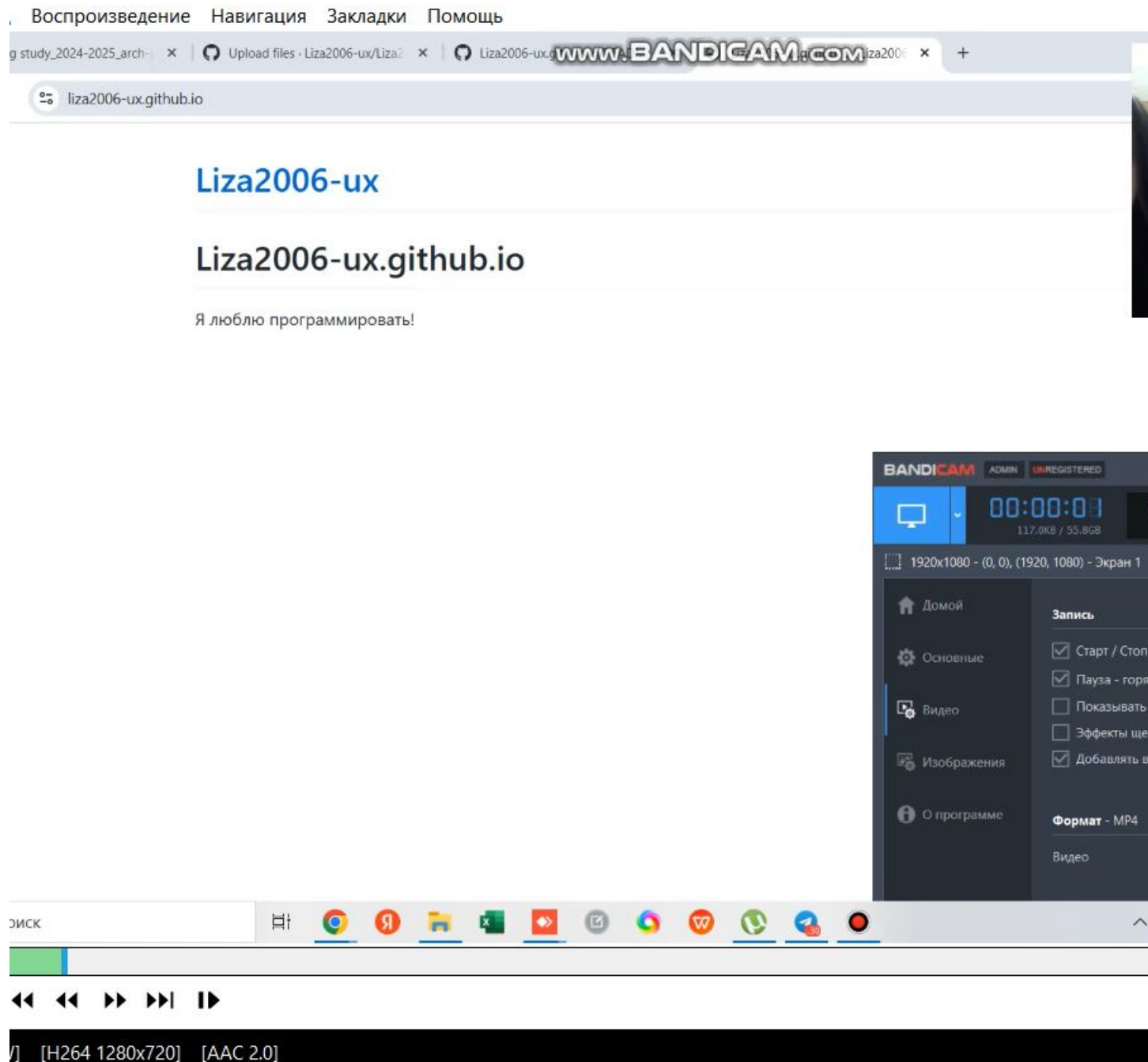
setting.

После этого выбрала в боковом меню GitHub setting>SSH-ключи и нажала кнопку Добавить ключ. Скопировав из локальной консоли ключ в буфер обмена:

```
cat ~/.ssh/id_rsa.pub | xclip -sel clip
```

Вставила ключ в появившееся на сайте поле.

После этого можно создала на сайте репозиторий, выбрав в меню , дала ему название и сделала общедоступным (публичным).
im 2025-03-04 21-57-55-498.mp4



Для загрузки репозитория из локального каталога на сервер выполнила следующие команды:

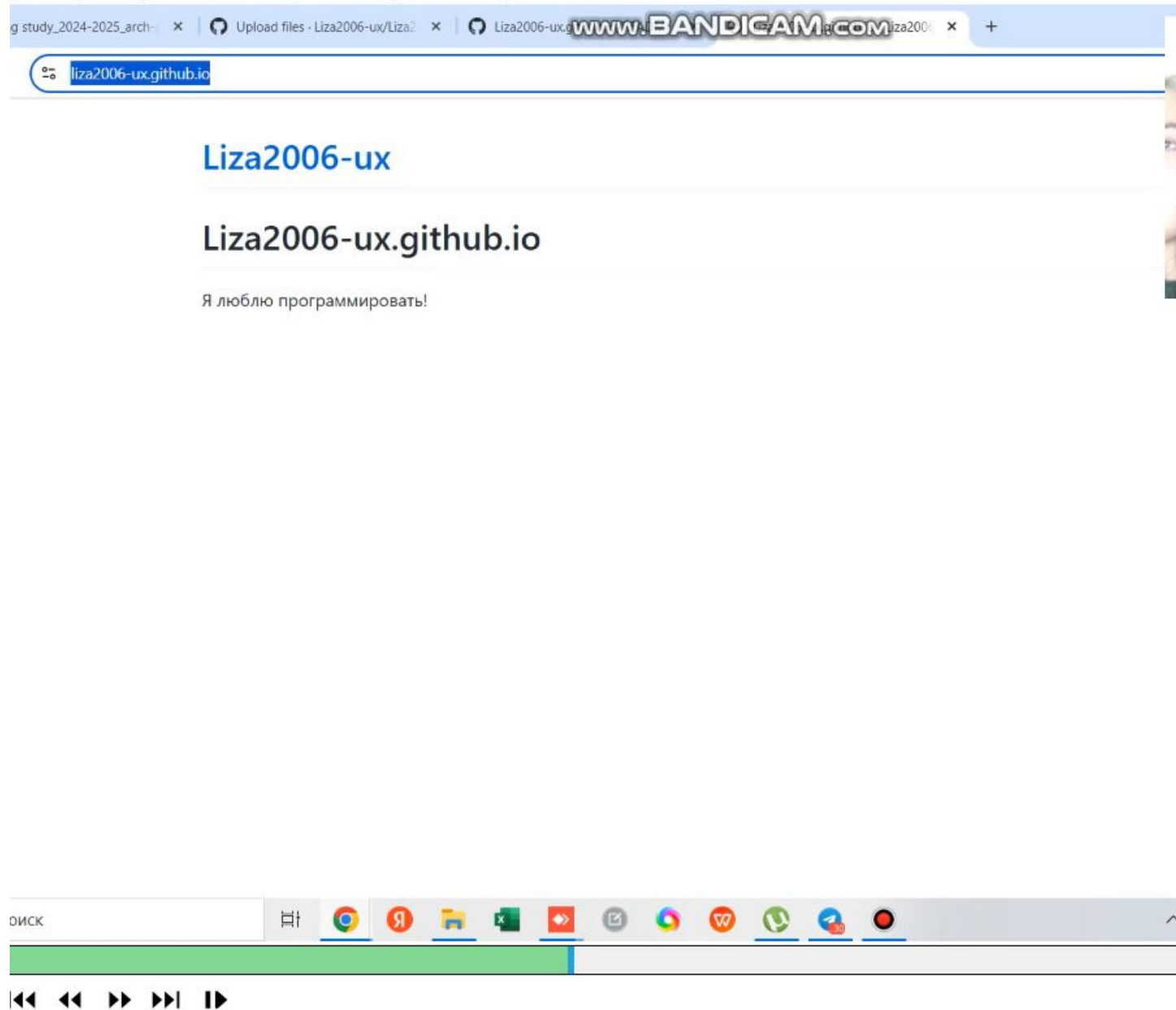
```
git remote add origin  
ssh://git@github.com/<username>/<reponame>.git  
git push -u origin master
```

Далее на локальном компьютере выполняла стандартные процедуры для работы с git при наличии центрального репозитория.

Базовая настройка git

Первичная настройка параметров git

Задала Лиза и email владельца репозитория:



```
git config --global user.name "Name Surname"
```

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

Настроила utf-8 в выводе сообщений git:

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

Настроила верификацию и подписание коммитов git.

Задала имя начальной ветки (будем называть её master):

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

Учёт переносов строк

В разных операционных системах приняты разные символы для перевода строк:

Windows: \r\n (CR и LF);

Unix: \n (LF);

Mac: \r (CR).

Посмотрела значения переносов строк в репозитории можно командой:

```
git ls-files --eol
```

Параметр autocrlf

Настройка core.autocrlf предназначена для того, чтобы в главном репозитории все переводы строк текстовых файлах были одинаковы.

Настройка core.autocrlf с параметрами true и input делает все переводы строк текстовых файлов в главном репозитории одинаковыми.

core.autocrlf true: конвертация CRLF->LF при коммите и обратно LF->CRLF при выгрузке кода из репозитория на файловую систему (обычно используется в Windows).

core.autocrlf input: конвертация CRLF->LF только при коммитах (используются в MacOS/Linux).

Варианты конвертации

The screenshot shows a Windows desktop environment. In the background, a web browser window displays the GitHub repository page for `Liza2006-ux`. The repository name is `Liza2006-ux.github.io`, and the file being viewed is `README.md`. The page content includes the repository name and the text "Я люблю программировать!". The browser's address bar shows the URL `github.com/Liza2006-ux/Liza2006-ux.github.io/blob/main/README.md`. The Windows taskbar at the bottom features a search bar and several application icons, including Chrome, Yandex, File Explorer, and others. In the foreground, the Bandicam video recording interface is visible on the right side of the screen. It shows a recording timer at `00:00:18`, a resolution of `1920x1080`, and a list of recording settings on the right, such as `Запись` (Recording) and `Формат` (Format).

Таблица 1.: Варианты конвертации для разных значений параметра `core.autocrlf`

<code>core.autocrlf</code>	<code>false</code>	<code>input</code>	<code>true</code>
<code>git commit</code>	<code>LF -> LF</code>	<code>LF -> LF</code>	<code>LF -> CRLF</code>
<code>CR -> CR</code>	<code>CR -> CR</code>		<code>CR -> CR</code>
<code>CRLF -> CRLF</code>	<code>CRLF -> LF</code>		<code>CRLF -> CRLF</code>
<code>git checkout</code>	<code>LF -> LF</code>	<code>LF -> LF</code>	<code>LF -> CRLF</code>
<code>CR -> CR</code>	<code>CR -> CR</code>		<code>CR -> CR</code>
<code>CRLF -> CRLF</code>	<code>CRLF -> CRLF</code>		<code>CRLF -> CRLF</code>

Установка параметра:

Для Windows

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

Для Linux

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

Параметр safecrlf

Установка параметра:

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

Создание ключа ssh

Общая информация

Алгоритмы шифрования ssh

Аутентификация

В SSH поддерживаются четыре алгоритма аутентификации по открытым ключам:

DSA:

размер ключей DSA не может превышать 1024, его следует отключить;

RSA:

следует создавать ключ большого размера: 4096 бит;

ECDSA:

ECDSA завязан на технологиях NIST, его следует отключить;

Ed25519:

используется пока не везде.

Симметричные шифры

Из 15 поддерживаемых в SSH алгоритмов симметричного шифрования, безопасными можно считать:

chacha20-poly1305;

aes*-ctr;

aes*-gcm.

Шифры 3des-cbc и arcfour потенциально уязвимы в силу использования DES и RC4.

Шифр cast128-cbc применяет слишком короткий размер блока (64 бит).

Обмен ключами

Применяемые в SSH методы обмена ключей DH (Diffie-Hellman) и ECDH (Elliptic Curve Diffie-Hellman) можно считать безопасными.

Из 8 поддерживаемых в SSH протоколов обмена ключами вызывают подозрения три, основанные на рекомендациях NIST:

ecdh-sha2-nistp256;

ecdh-sha2-nistp384;

ecdh-sha2-nistp521.

Не стоит использовать протоколы, основанные на SHA1.

Файлы ssh-ключей

По умолчанию пользовательские ssh-ключи сохраняются в каталоге ~/.ssh в домашнем каталоге пользователя.

Убедитесь, что у вас ещё нет ключа.

Файлы закрытых ключей имеют названия типа `id_<алгоритм>` (например, `id_dsa`, `id_rsa`).

По умолчанию закрытые ключи имеют имена:

`id_dsa`
`id_ecdsa`
`id_ed25519`
`id_rsa`

Открытые ключи имеют дополнительные расширения `.pub`.

По умолчанию публичные ключи имеют имена:

`id_dsa.pub`
`id_ecdsa.pub`
`id_ed25519.pub`
`id_rsa.pub`

При создании ключа команда попросит ввести любую ключевую фразу для более надёжной защиты вашего пароля. Можно пропустить этот этап, нажав Enter.

Сменить пароль на ключ можно с помощью команды:

```
ssh-keygen -p
```

Создание ключа ssh

Ключ ssh создаётся командой:

```
ssh-keygen -t <алгоритм>
```

Создайте ключи:

по алгоритму `rsa` с ключём размером 4096 бит:

```
ssh-keygen -t rsa -b 4096
```

по алгоритму `ed25519`:

```
ssh-keygen -t ed25519
```

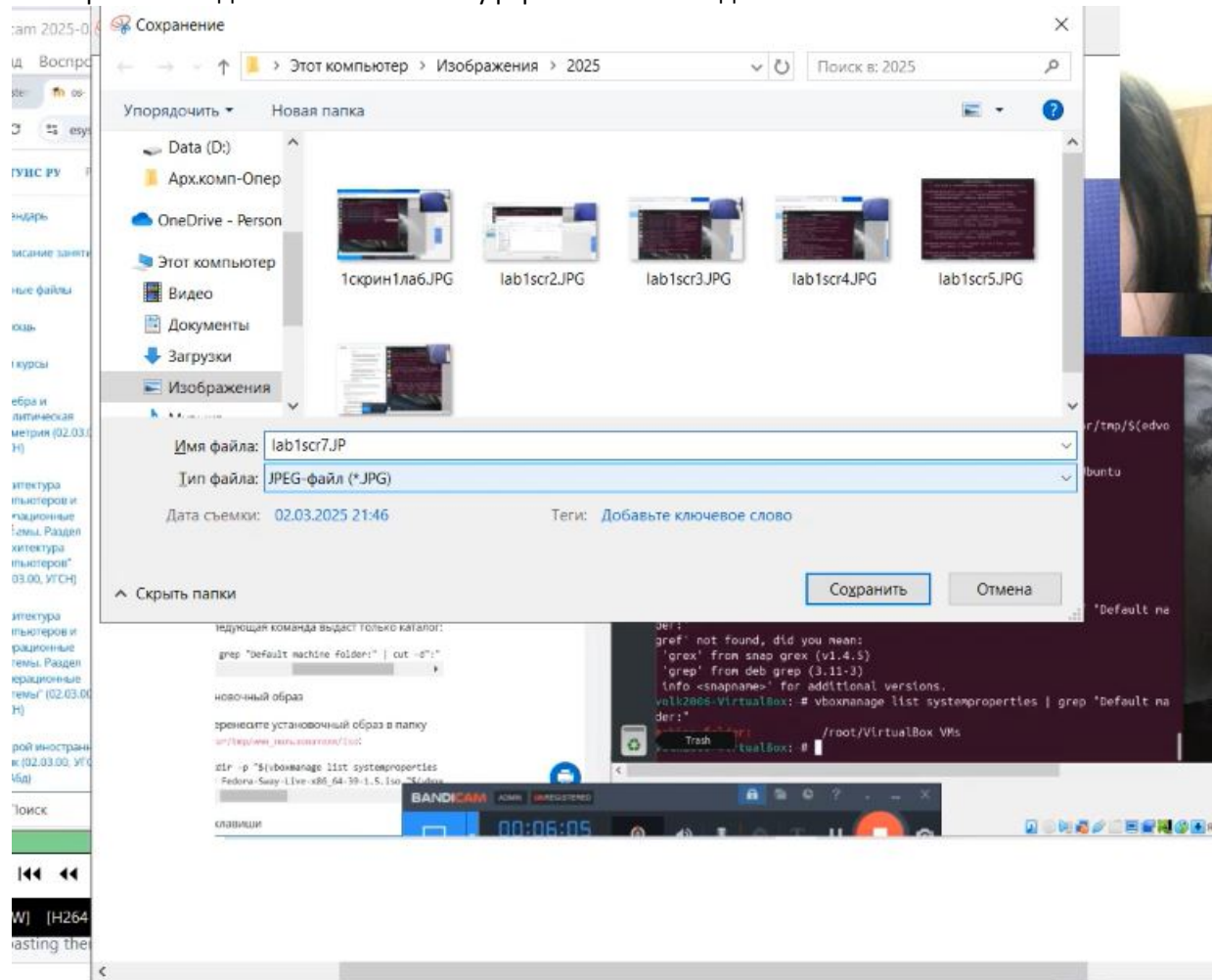
При создании ключа команда попросит ввести любую ключевую фразу для более надёжной защиты вашего пароля. Можно пропустить этот этап, нажав Enter.

Сменила пароль на ключ можно с помощью команды:

```
ssh-keygen -p
```

Добавление SSH-ключа в учётную запись GitHub

Скопировала созданный SSH-ключ в буфер обмена командой:



```
xclip -i < ~/.ssh/id_ed25519.pub
```

Открыла настройки своего аккаунта на GitHub и перешла в раздел SSH and GPG keys.

Нажала кнопку New SSH key.

Добавила в поле Title название этого ключа, например, ed25519@hostname.

Вставила из буфера обмена в поле Key ключ.

Нажала кнопку Add SSH key.

Верификация коммитов с помощью PGP

Как настроить PGP-подпись коммитов с помощью gpg.

Общая информация

Коммиты имеют следующие свойства:

author (автор) — контрибьютор, выполнивший работу (указывается для справки);

committer (коммитер) — пользователь, который закоммитил изменения.

Эти свойства можно переопределить при совершении коммита.

Авторство коммита можно подделать.

В git есть функция подписи коммитов.

Для подписывания коммитов используется технология PGP (см. Работа с PGP).
Подпись коммита позволяет удостовериться в том, кто является коммитером. Авторство не проверяется.

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.

Последовательность выполнения работы

Установка программного обеспечения

Установка git
Установила git:

```
dnf install git
```

Установка gh
Fedora:

```
dnf install gh
```

Базовая настройка git
Задала имя и email владельца репозитория:

```
git config --global user.name "Name Surname"
```

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

Настроим utf-8 в выводе сообщений git:

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

Настроила верификацию и подписание коммитов git (см. Верификация коммитов git с помощью GPG).

Задала 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
```

Создала ключи ssh

по алгоритму rsa с ключём размером 4096 бит:

```
ssh-keygen -t rsa -b 4096
```

по алгоритму ed25519:

```
ssh-keygen -t ed25519
```

Создала ключи pgp

Генерируем ключ

```
gpg --full-generate-key
```

Из предложенных опций выбираем:

тип RSA and RSA;

размер 4096;

выбираем срок действия; значение по умолчанию — 0 (срок действия не истекает никогда).

GPG запросит личную информацию, которая сохранится в ключе:

Имя (Елизавета).

Адрес электронной почты.

При вводе email убедитесь, что он соответствует адресу, используемому на GitHub.

Комментарий

Настройка github

Создайте учётную запись на <https://github.com>.

Заполните основные данные на <https://github.com>.

Добавление PGP ключа в GitHub

Вывела список ключей и скопировала 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

Скопировала ваш сгенерированный PGP ключ в буфер обмена:

```
gpg --armor --export <pgp fingerprint> | xclip -sel clip
```

Перешла в настройки GitHub (<https://github.com/settings/keys>), нажала на кнопку New GPG key и вставьте полученный ключ в поле ввода.

Настройка автоматических подписей коммитов git

Используя введённый email, указала Git применять его при подписи коммитов:

```
git config --global user.signingkey <pgp fingerprint>
```

```
git config --global commit.gpgsign true
```

```
git config --global gpg.program $(which gpg2)
```

Настройка gh

Для начала необходимо авторизоваться

```
gh auth login
```

Утилита задала несколько наводящих вопросов.

Авторизовалась через браузер.

Шаблон для рабочего пространства

Рабочее пространство для лабораторной работы

Репозиторий: <https://github.com/yamadharma/course-directory-student-template>.

Создание репозитория курса на основе шаблона

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:<owner>/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
```