

# **Отчёт по лабораторной работе №2**

**Первоначальная настройка git**

**Тагиев Павел Фаикович**

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
4.1	Подготовительные действия на WSL + Ubuntu . . . . .	7
4.2	Конфигурация git . . . . .	8
4.3	Генерация ssh ключа . . . . .	9
4.4	Добавление публичного ssh ключа на github . . . . .	10
4.5	Генерация gpg ключа . . . . .	12
4.6	Автоматическое подписание коммитов и добавление ключа на github . . . . .	14
4.7	Установка утилиты gh и авторизация . . . . .	15
4.8	Создание репозитория курса из шаблона . . . . .	18
<b>5</b>	<b>Ответы на контрольные вопросы</b>	<b>21</b>
<b>6</b>	<b>Выводы</b>	<b>24</b>
	<b>Список литературы</b>	<b>25</b>

## Список иллюстраций

4.1	Мой .bashrc . . . . .	8
4.2	Конфигурация git . . . . .	8
4.3	Генерация ssh ключей . . . . .	9
4.4	Настройки github . . . . .	10
4.5	Добавленный публичный ключ . . . . .	11
4.6	Проверка ключа . . . . .	11
4.7	Генерация gpg ключа . . . . .	13
4.8	Настройка автоматического подписания коммитов . . . . .	14
4.9	Добавление gpg ключа на github . . . . .	15
4.10	Установка gh . . . . .	16
4.11	Авторизация на github . . . . .	17
4.12	Сообщение об успешной авторизации . . . . .	17
4.13	Создания репозитория по шаблону . . . . .	18
4.14	Создание структуры курса . . . . .	19
4.15	Изменения на удаленном репозитории . . . . .	20
4.16	Подписанные коммиты . . . . .	20

# 1 Цель работы

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

## 2 Задание

- Произвести базовую конфигурацию git.
- Создать ssh ключ и добавить его на github
- Создать gpg ключ и добавить его на github. Настроить автоматическое подписание коммитов.
- Создать репозиторий курса на основе шаблона.

## 3 Теоретическое введение

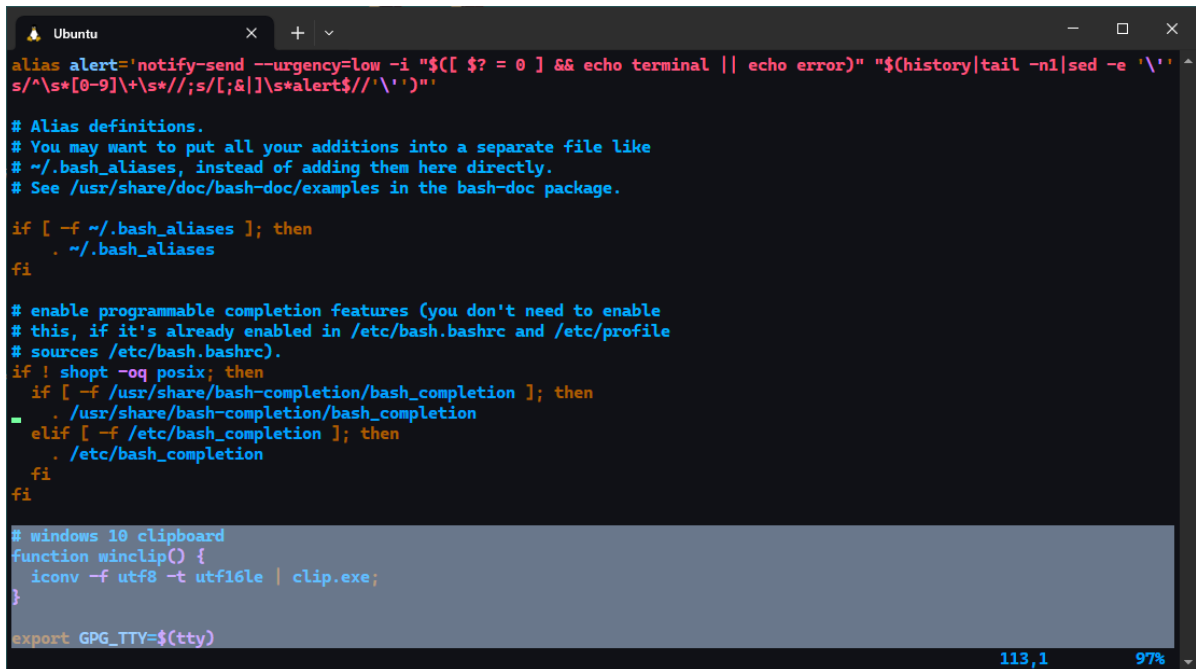
Git — система управления версиями с распределенной архитектурой. В отличие от некогда популярных систем вроде CVS и Subversio (SVN), где полная история версий проекта доступна лишь в одном месте, в Git каждая рабочая копия кода сама по себе является репозиторием. Это позволяет всем разработчикам хранить историю изменений в полном объеме.

Разработка в Git ориентирована на обеспечение высокой производительности, безопасности и гибкости распределенной системы [1].

## 4 Выполнение лабораторной работы

### 4.1 Подготовительные действия на WSL + Ubuntu

Так как лабораторная работа №2 мной выполнялась на WSL [2] с установленной ОС Ubuntu, нужно произвести некоторые подготовительные действия. Добавим в файл `~/.bashrc`, строку `export GPG_TTY=$(tty)`, она нужна чтобы при использовании `gpg` ключа кодовое слово спрашивалось в текущем терминале. Еще нам будет полезна функция, которую я назвал `winclip` (рис. 4.1). Переданная ей через пайп строка преобразуется из `utf8` в `utf16le` и затем копируется в буфер обмена Windows. Например, после выполнения `echo "Привет из wsl!" | winclip` в буфер обмена Windows будет скопирована строка `Привет из wsl!`. Далее я буду использовать эту функцию для копирования публичных ключей `ssh` и `gpg`.



```
alias alert='notify-send --urgency=low -i "${[ $? = 0 ] && echo terminal || echo error}" "$(history|tail -n1|sed -e '\''
s/^s*[0-9]\+\s*//;s/[:&]\s*alert$//'\''")'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

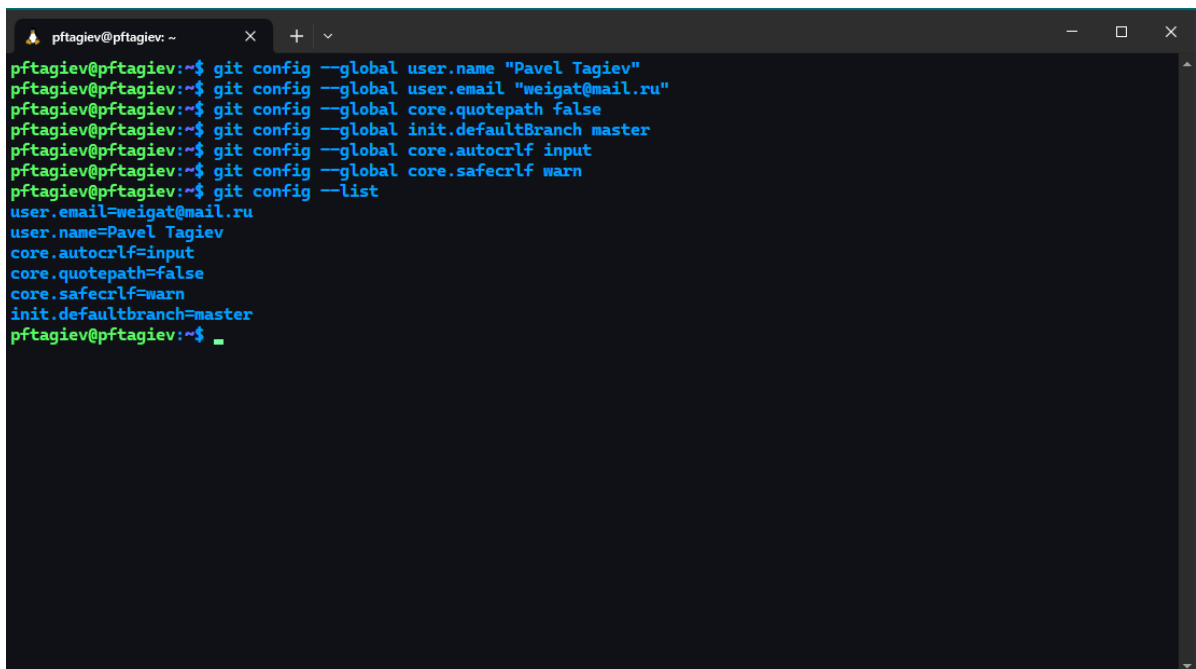
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

# windows 10 clipboard
function winclip() {
    iconv -f utf8 -t utf16le | clip.exe;
}

export GPG_TTY=$(tty)
```

Рис. 4.1: Мой .bashrc

## 4.2 Конфигурация git



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

Рис. 4.2: Конфигурация git

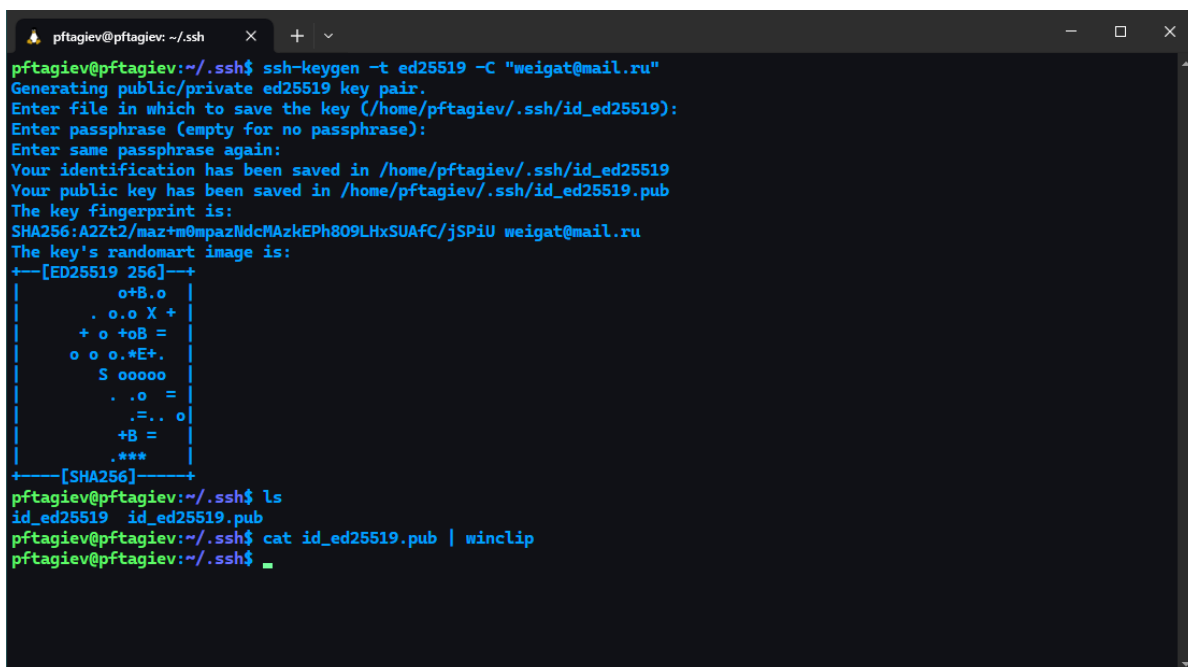


- В первых двух строках на рис. 4.2 мы глобально задаем свое имя и email.
- Третья строка включает utf8 при выводе сообщений git.
- Четвертая задает имя master дефолтной ветке.
- Пятая строка задает переносы строк для Linux.
- Шестая включает предупреждение о необратимом преобразовании переноса строк.

Далее выводится текущая конфигурация (рис. 4.2).

## 4.3 Генерация ssh ключа

Публичный и приватный ключ можно сгенерировать командой `ssh-keygen`. Пара ключей хранится в директории `~/ .ssh`. Сгенерируем наш ключ по алгоритму ed25519, оставим комментарий с помощью флага `-C`, в качестве комментария укажем нашу почту. Все шаги утилиты при генерации ключа опциональны, пропустим их нажимая клавишу **Enter**. Скопируем публичный ключ в буфер обмена Windows с помощью описанной ранее функции `winclip` (рис. 4.3).



```
pftagiev@pftagiev: ~/.ssh$ ssh-keygen -t ed25519 -C "weigat@mail.ru"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/pftagiev/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/pftagiev/.ssh/id_ed25519
Your public key has been saved in /home/pftagiev/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:A2Zt2/maz+m0mpazNdcMAzkEPH809LHxSUAfC/jSPiU weigat@mail.ru
The key's randomart image is:
+--[ED25519 256]--+
|      . o . o X + |
|    + o + o B =   |
|   o o o . * E + . |
|    S  ooooo      |
|      . o  =      |
|     . = . . o    |
|    + B =         |
|     .***         |
+---[SHA256]-----+
pftagiev@pftagiev: ~/.ssh$ ls
id_ed25519  id_ed25519.pub
pftagiev@pftagiev: ~/.ssh$ cat id_ed25519.pub | winclip
pftagiev@pftagiev: ~/.ssh$
```

Рис. 4.3: Генерация ssh ключей

## 4.4 Добавление публичного ssh ключа на github

Перейдем в настройки аккаунта github, в раздел **SSH and GPG keys** (рис. 4.4). Нажмем кнопку **New SSH key**, добавим скопированный ранее публичный ключ (рис. 4.5).

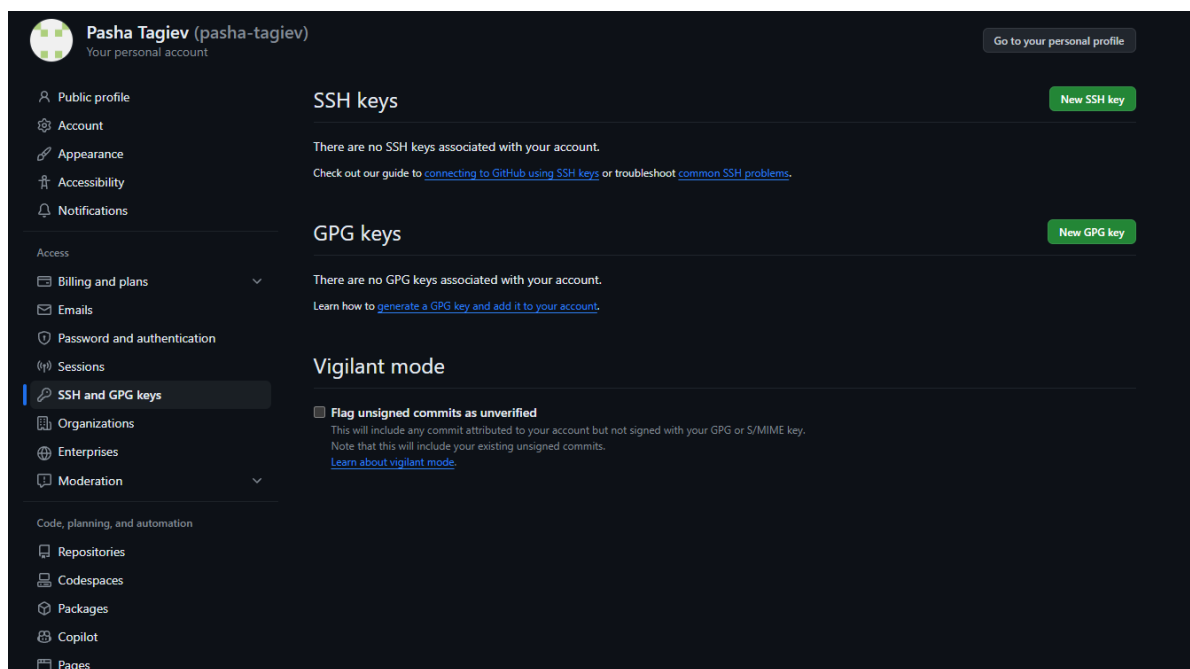


Рис. 4.4: Настройки github

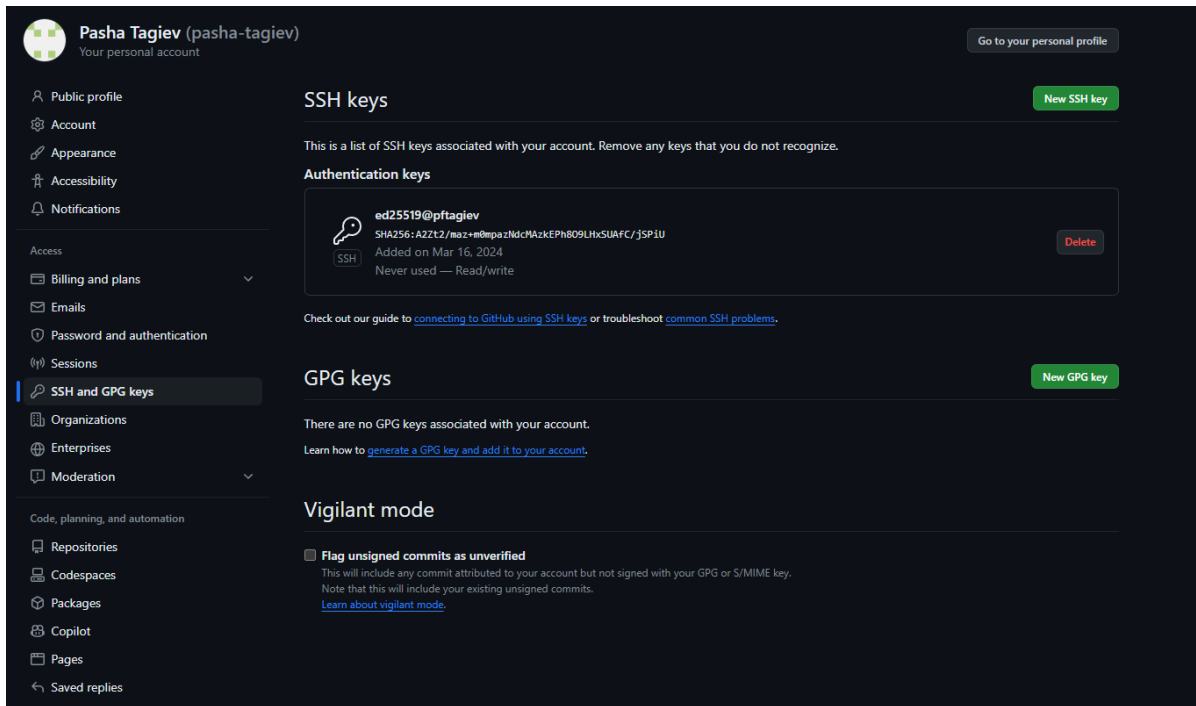


Рис. 4.5: Добавленный публичный ключ

Проверим наш ключ, попробовав подключиться к git по ssh (рис. 4.6). Как можно увидеть, ключ работает.

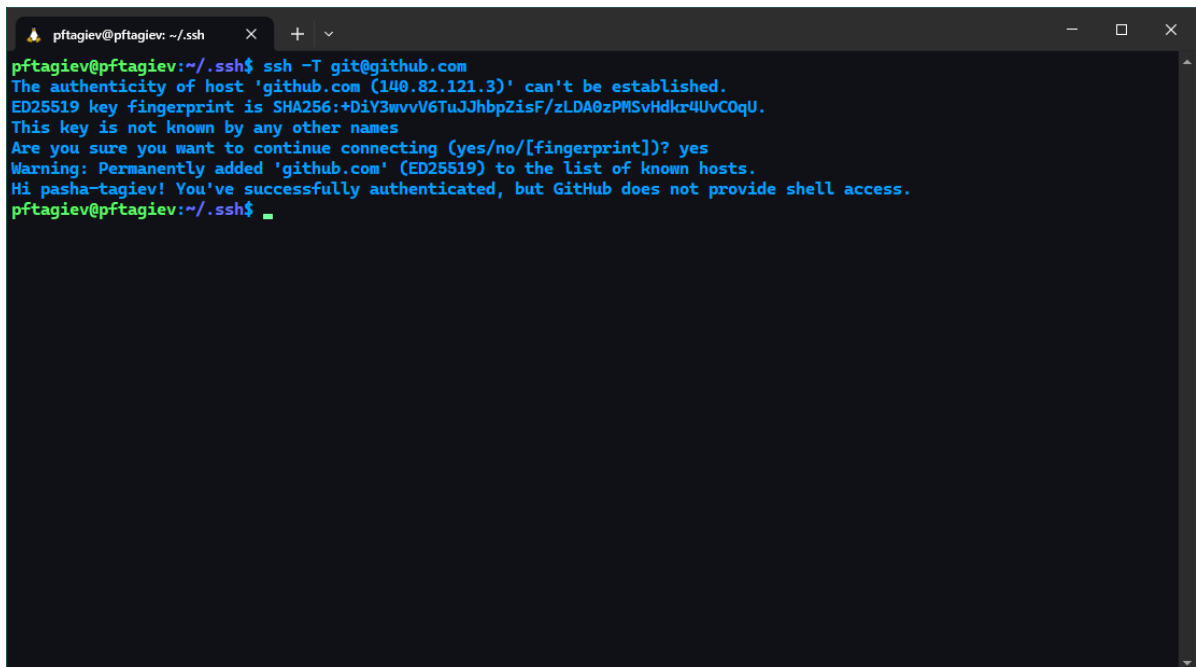


Рис. 4.6: Проверка ключа

## 4.5 Генерация gpg ключа

Сгенерируем gpg ключ. Для этого введем в терминале `gpg --full-generate-key`. Генерация ключей с помощью этой утилиты происходит в несколько этапов (рис. 4.7), опишем каждый из них:

1. Нас просят выбрать тип ключа. Выбираем RSA, введя в терминал цифру 1 и нажав **Enter**.
2. Далее просят указать размер ключа выбираем максимально возможный - 4096.
3. Тут нужно указать через какое время ключ станет недействительным. Укажем 0, что означает что у него нет срока годности. Повторно подтверждаем наш выбор введя у и нажав **Enter**.
4. Теперь нас просят ввести имя и email, email должен совпадать с тем что используется в github аккаунте.
5. Далее нужно ввести кодовую фразу.
6. После начнется создание ключа. Утилита попросит нас совершать как можно больше действий с машиной (набор текста на клавиатуре, перемещение курсора мыши и т.д.) во время создания ключа, это нужно для более качественной генерации случайных чисел.

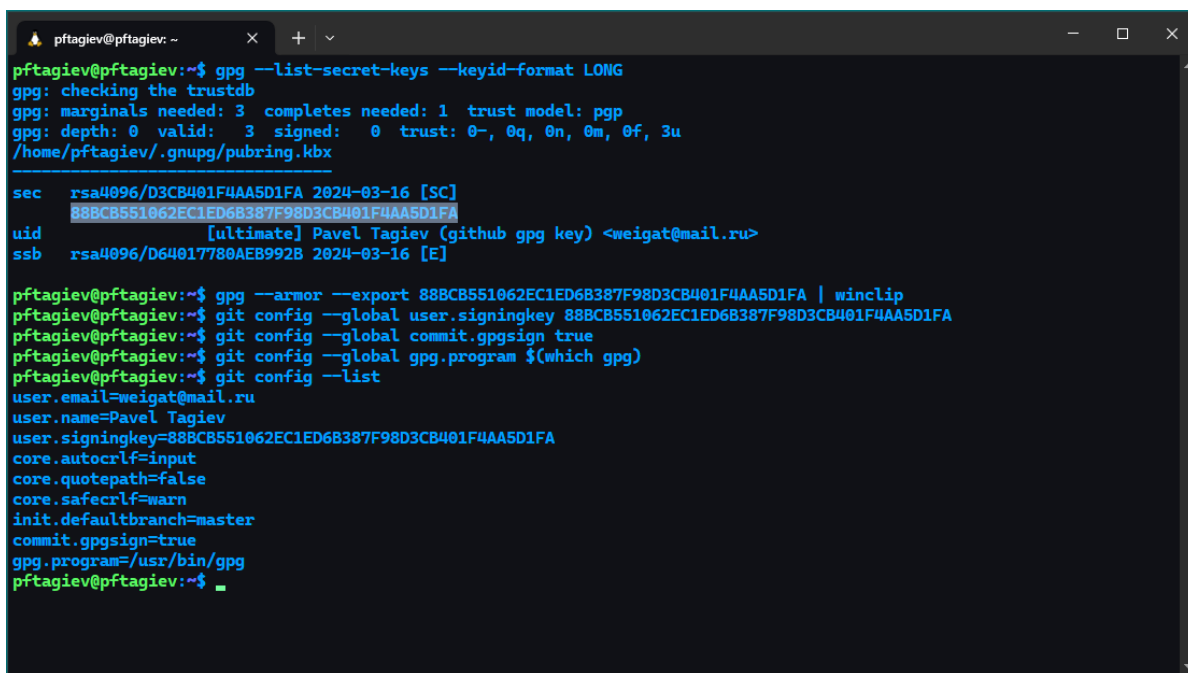
```
pftagiev@pftagiev: ~  
pftagiev@pftagiev:~$ gpg --full-generate-key  
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law.  
  
Please select what kind of key you want:  
  (1) RSA and RSA (default)  
  (2) DSA and Elgamal  
  (3) DSA (sign only)  
  (4) RSA (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: Pavel Tagiev  
Email address: weigat@mail.ru  
Comment: github gpg key  
You selected this USER-ID:  
  "Pavel Tagiev (github gpg key) <weigat@mail.ru>"  
  
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? 0  
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: key D3CB401F4AA5D1FA marked as ultimately trusted  
gpg: revocation certificate stored as '/home/pftagiev/.gnupg/openpgp-revocs.d/88B  
CB551062EC1ED6B387F98D3CB401F4AA5D1FA.rev'  
public and secret key created and signed.  
  
pub   rsa4096 2024-03-16 [SC]  
      88BCB551062EC1ED6B387F98D3CB401F4AA5D1FA  
uid           Pavel Tagiev (github gpg key) <weigat@mail.ru>  
sub   rsa4096 2024-03-16 [E]
```

Рис. 4.7: Генерация gpg ключа

## 4.6 Автоматическое подписание коммитов и добавление ключа на github

На рис. 4.8 можно увидеть настройку автоматического подписания коммитов, разберем каждый шаг:

1. Выводим список gpg ключей, копируем id нужного нам ключа.
2. Записываем публичный ключ в буфер обмена Windows.
3. Добавляем id ключа в глобальный конфиг git.
4. Включаем подписание коммитов.
5. Указываем программу для генерации gpg в глобальном конфиге git.



```
pftagiev@pftagiev: ~  
pftagiev@pftagiev:~$ gpg --list-secret-keys --keyid-format LONG  
gpg: checking the trustdb  
gpg: marginals needed: 3 completes needed: 1 trust model: pgp  
gpg: depth: 0 valid: 3 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 3u  
/home/pftagiev/.gnupg/pubring.kbx  
  
sec  rsa4096/D3CB401F4AA5D1FA 2024-03-16 [SC]  
      888CB551062EC1ED6B387F98D3CB401F4AA5D1FA  
uid          [ultimate] Pavel Tagiev (github gpg key) <weigat@mail.ru>  
ssb  rsa4096/D64017780AEB992B 2024-03-16 [E]  
  
pftagiev@pftagiev:~$ gpg --armor --export 888CB551062EC1ED6B387F98D3CB401F4AA5D1FA | winclip  
pftagiev@pftagiev:~$ git config --global user.signingkey 888CB551062EC1ED6B387F98D3CB401F4AA5D1FA  
pftagiev@pftagiev:~$ git config --global commit.gpgsign true  
pftagiev@pftagiev:~$ git config --global gpg.program $(which gpg)  
pftagiev@pftagiev:~$ git config --list  
user.email=weigat@mail.ru  
user.name=Pavel Tagiev  
user.signingkey=888CB551062EC1ED6B387F98D3CB401F4AA5D1FA  
core.autocrlf=input  
core.quotepath=false  
core.safecrlf=warn  
init.defaultbranch=master  
commit.gpgsign=true  
gpg.program=/usr/bin/gpg  
pftagiev@pftagiev:~$
```

Рис. 4.8: Настройка автоматического подписания коммитов

Добавим скопированный ранее публичный ключ в настройки github по аналогии с ssh ключем. Включим **Vigilant mode** (рис. 4.9). Для отображения подписанных и неподписанных коммитов на github.

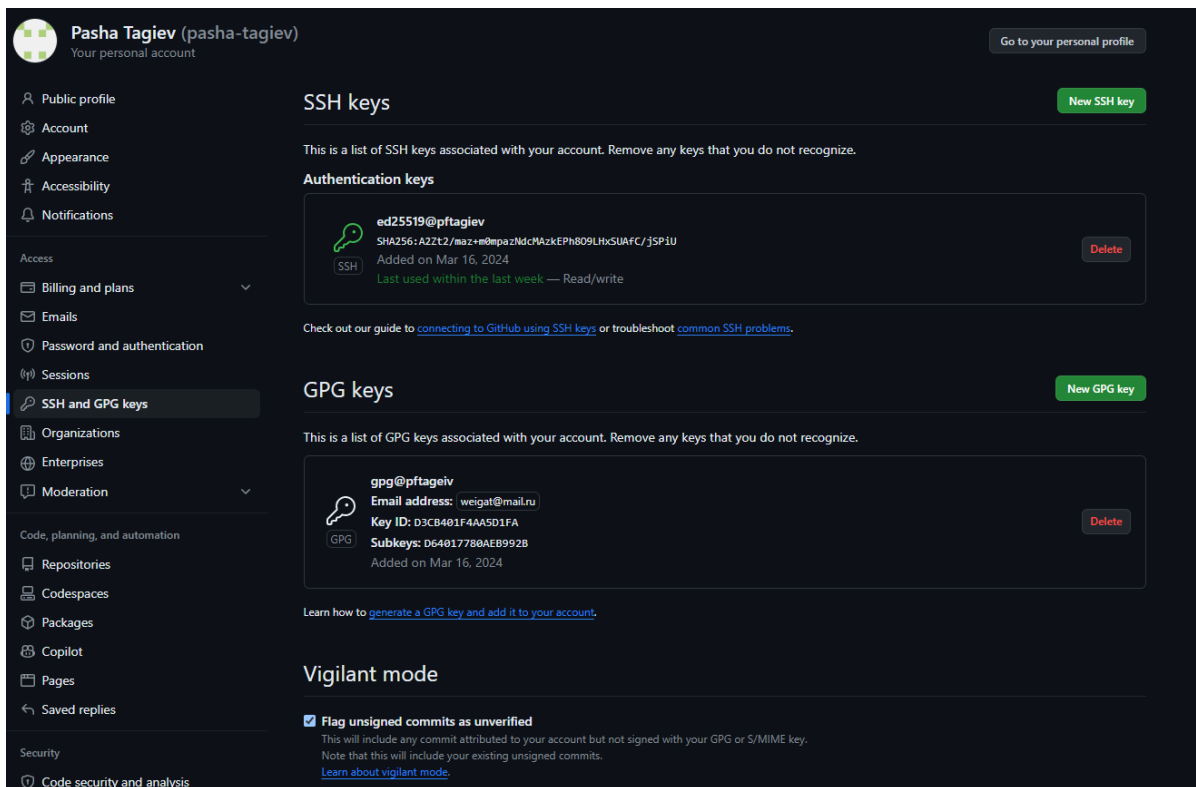
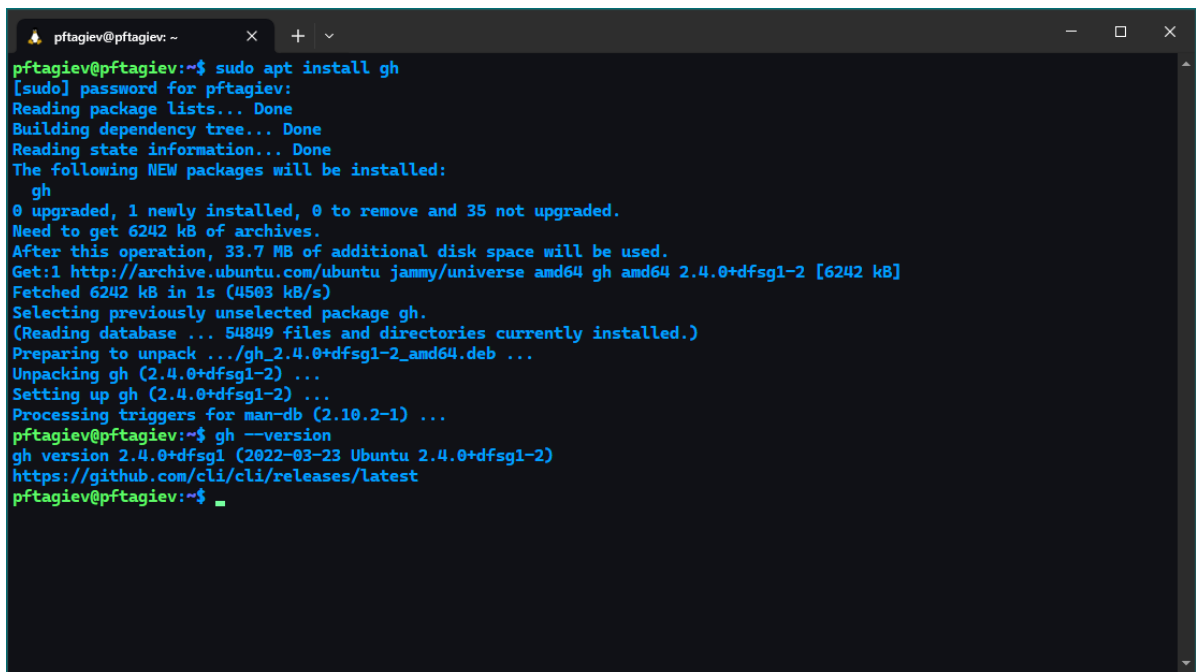


Рис. 4.9: Добавление gpg ключа на github

## 4.7 Установка утилиты gh и авторизация

По причине того, что у меня уже есть git на WSL, повтроно я его устанавливать не буду, но на Ubuntu это можно сделать следующей командой `sudo apt install git`. Установку gh можно увидеть на рис. 4.10.



```
pftagiev@pftagiev: ~  
pftagiev@pftagiev:~$ sudo apt install gh  
[sudo] password for pftagiev:  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
The following NEW packages will be installed:  
  gh  
0 upgraded, 1 newly installed, 0 to remove and 35 not upgraded.  
Need to get 6242 kB of archives.  
After this operation, 33.7 MB of additional disk space will be used.  
Get:1 http://archive.ubuntu.com/ubuntu jammy/universe amd64 gh amd64 2.4.0+dfsg1-2 [6242 kB]  
Fetched 6242 kB in 1s (4503 kB/s)  
Selecting previously unselected package gh.  
(Reading database ... 54849 files and directories currently installed.)  
Preparing to unpack .../gh_2.4.0+dfsg1-2_amd64.deb ...  
Unpacking gh (2.4.0+dfsg1-2) ...  
Setting up gh (2.4.0+dfsg1-2) ...  
Processing triggers for man-db (2.10.2-1) ...  
pftagiev@pftagiev:~$ gh --version  
gh version 2.4.0+dfsg1 (2022-03-23 Ubuntu 2.4.0+dfsg1-2)  
https://github.com/cli/cli/releases/latest  
pftagiev@pftagiev:~$
```

Рис. 4.10: Установка gh

Авторизируемся на github с помощью gh (рис. 4.11). Я выбрал авторизацию через браузер, но так как на WSL он у меня не установлен. Я открыл ссылку из терминал в браузере на Windows и вставил код. После ответа на несколько вопросов мне удалось авторизоваться на github (рис. 4.12).



```
pftagiev@pftagiev: ~  
pftagiev@pftagiev:~$ gh auth login  
? What account do you want to log into? GitHub.com  
? What is your preferred protocol for Git operations? SSH  
? Upload your SSH public key to your GitHub account? Skip  
? How would you like to authenticate GitHub CLI? Login with a web browser  
  
! First copy your one-time code: 71C9-B2ED  
- Press Enter to open github.com in your browser...  
/usr/bin/xdg-open: 882: x-www-browser: not found  
/usr/bin/xdg-open: 882: firefox: not found  
/usr/bin/xdg-open: 882: iceweasel: not found  
/usr/bin/xdg-open: 882: seamonkey: not found  
/usr/bin/xdg-open: 882: mozilla: not found  
/usr/bin/xdg-open: 882: epiphany: not found  
/usr/bin/xdg-open: 882: konqueror: not found  
/usr/bin/xdg-open: 882: chromium: not found  
/usr/bin/xdg-open: 882: chromium-browser: not found  
/usr/bin/xdg-open: 882: google-chrome: not found  
/usr/bin/xdg-open: 882: www-browser: not found  
/usr/bin/xdg-open: 882: links2: not found  
/usr/bin/xdg-open: 882: elinks: not found  
/usr/bin/xdg-open: 882: links: not found  
/usr/bin/xdg-open: 882: lynx: not found  
/usr/bin/xdg-open: 882: w3m: not found  
xdg-open: no method available for opening 'https://github.com/login/device'  
! Failed opening a web browser at https://github.com/login/device  
exit status 3  
Please try entering the URL in your browser manually  
  
✓ Authentication complete. Press Enter to continue...  
- gh config set -h github.com git_protocol ssh  
✓ Configured git protocol  
✓ Logged in as pasha-tagiev
```

Рис. 4.11: Авторизация на github

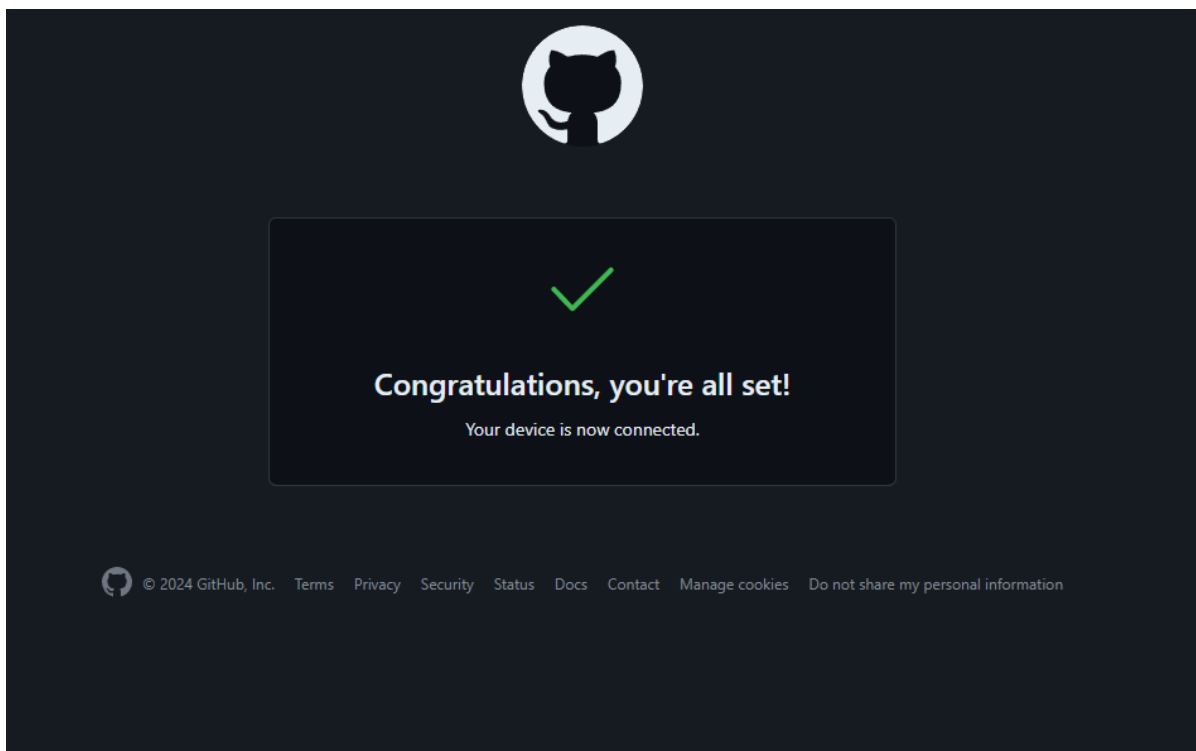


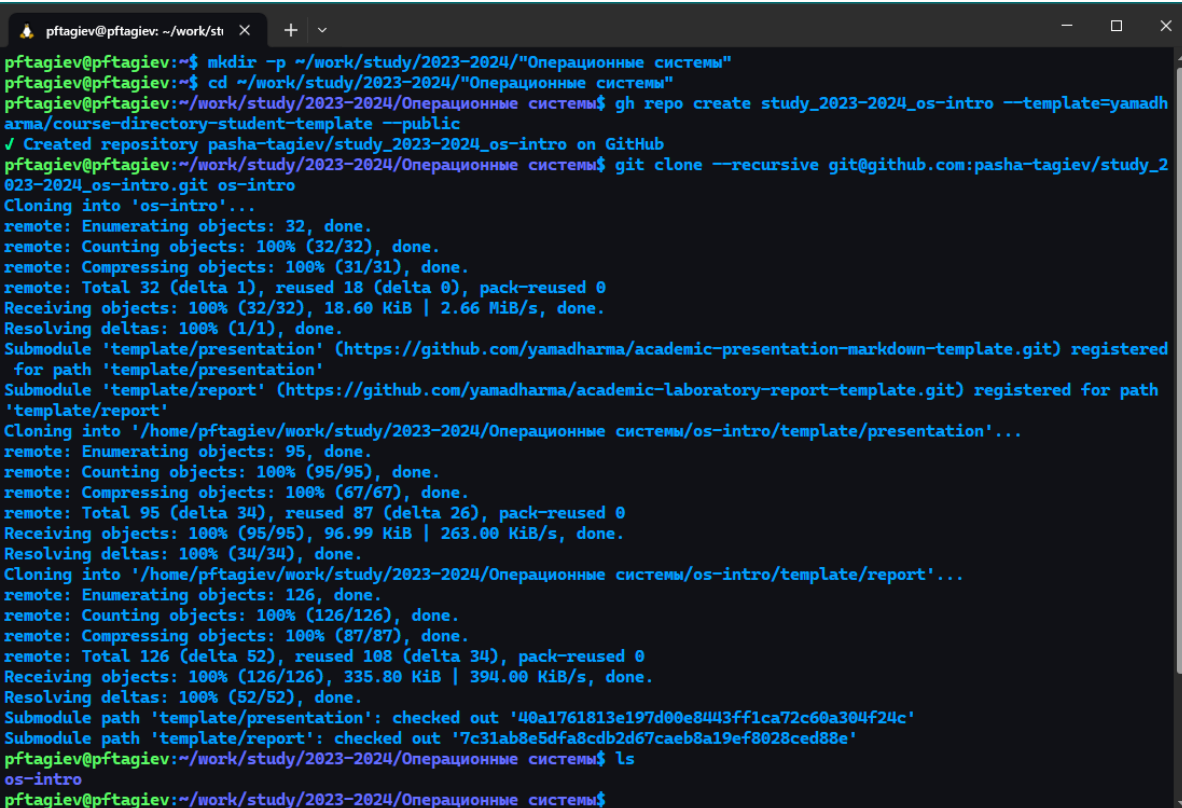
Рис. 4.12: Сообщение об успешной авторизации

## 4.8 Создание репозитория курса из шаблона

Создадим директорию для выполнения лабораторных работ командой `mkdir -p <директория>`. И перейдем в нее командой `cd`. Создаем публичный удаленный репозиторий с помощью `gh`, по шаблону:

```
yamadharm/course-directory-student-template
```

Клонируем созданный удаленный репозиторий в папку `os-intro` с флагом `--recursive`, для клонирования и его подмодулей (рис. 4.13).



```
pftagiev@pftagiev: ~/work/sti
pftagiev@pftagiev:~$ mkdir -p ~/work/study/2023-2024/"Операционные системы"
pftagiev@pftagiev:~$ cd ~/work/study/2023-2024/"Операционные системы"
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы$ gh repo create study_2023-2024_os-intro --template=yamadharm/course-directory-student-template --public
✓ Created repository pasha-tagiev/study_2023-2024_os-intro on GitHub
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы$ git clone --recursive git@github.com:pasha-tagiev/study_2023-2024_os-intro.git os-intro
Cloning into '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
Receiving objects: 100% (32/32), 18.60 KiB | 2.66 MiB/s, done.
Resolving deltas: 100% (1/1), done.
Submodule 'template/presentation' (https://github.com/yamadharm/academic-presentation-markdown-template.git) registered for path 'template/presentation'
Submodule 'template/report' (https://github.com/yamadharm/academic-laboratory-report-template.git) registered for path 'template/report'
Cloning into '/home/pftagiev/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
Receiving objects: 100% (95/95), 96.99 KiB | 263.00 KiB/s, done.
Resolving deltas: 100% (34/34), done.
Cloning into '/home/pftagiev/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
Receiving objects: 100% (126/126), 335.80 KiB | 394.00 KiB/s, done.
Resolving deltas: 100% (52/52), done.
Submodule path 'template/presentation': checked out '40a1761813e197d00e8443ff1ca72c60a304f24c'
Submodule path 'template/report': checked out '7c31ab8e5dfa8cdb2d67caeb8a19ef8028ced88e'
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы$ ls
os-intro
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы$
```

Рис. 4.13: Создания репозитория по шаблону

Переходим в папку `os-intro`, удаляем лишний файл `package.json`. Записываем в файл `COURSE` название курса (в нашем случае `os-intro`). Вызываем `make` для таргета `submodule`, чтобы обновить подмодули, затем `make prepare`, чтобы сгенерировать структуру курса (рис. 4.14).

```
pftagiev@pftagiev: ~/work/sti X + v
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы$ cd os-intro/
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы/os-intro$ rm package.json
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы/os-intro$ ls
CHANGELOG.md  COURSE  LICENSE  Makefile  README.en.md  README.git-flow.md  README.md  config  template
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы/os-intro$ echo os-intro > COURSE
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы/os-intro$ cat COURSE
os-intro
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы/os-intro$ make
Usage:
  make <target>

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

pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы/os-intro$ make submodule
git submodule update --init --recursive
git submodule foreach 'git fetch origin; git checkout $(git rev-parse --abbrev-ref HEAD); git reset --hard origin/$(git
rev-parse --abbrev-ref HEAD); git submodule update --recursive; git clean -dfx'
Entering 'template/presentation'
HEAD is now at 40a1761 Merge branch 'release/1.0.3'
Entering 'template/report'
HEAD is now at 7c31ab8 Merge branch 'release/1.0.4'
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы/os-intro$ make prepare
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы/os-intro$ ls labs/
README.md  lab01  lab03  lab05  lab07  lab09  lab11  lab13  lab15
README.ru.md  lab02  lab04  lab06  lab08  lab10  lab12  lab14
pftagiev@pftagiev:~/work/study/2023-2024/Операционные системы/os-intro$
```

Рис. 4.14: Создание структуры курса

Добавляем файлы в индекс git командой `git add .`, фиксируем изменения с требуемым по заданию комментарием:

```
git commit -m "feat(main): make course structure"
```

Нас попросят ввести кодовую фразу, которую мы вводили при генерации gpg ключа. После успешного ввода фразы отправляем изменения в удаленный репозиторий `git push` (рис. 4.15, 4.16).

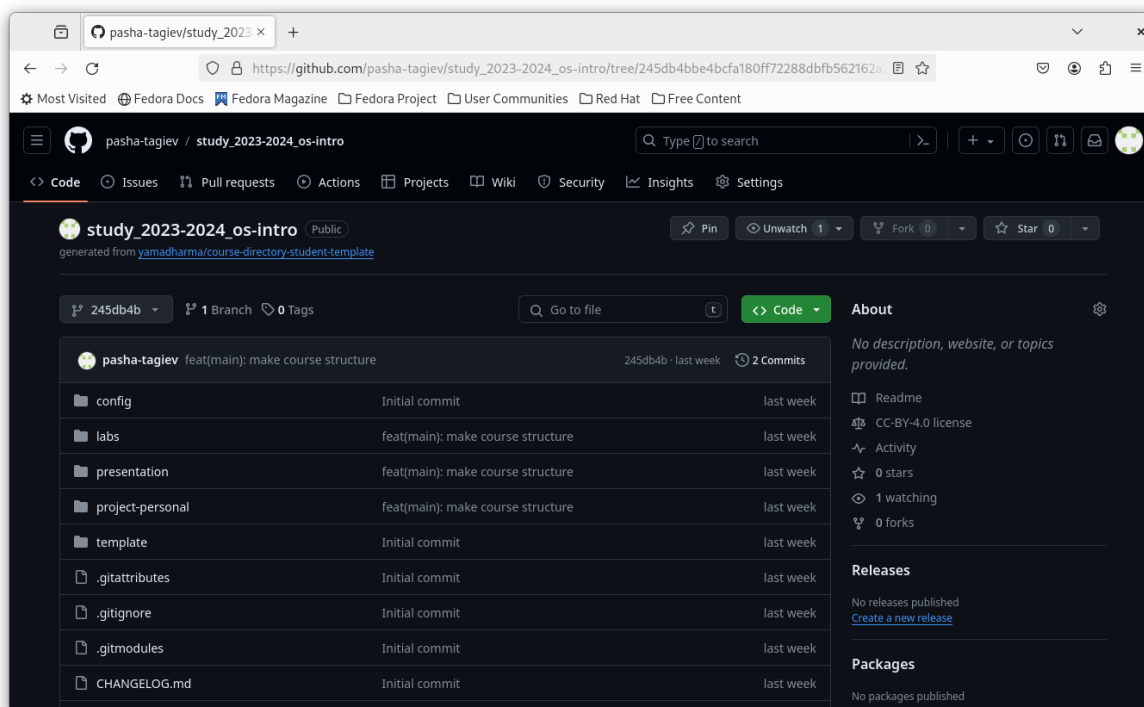


Рис. 4.15: Изменения на удаленном репозитории

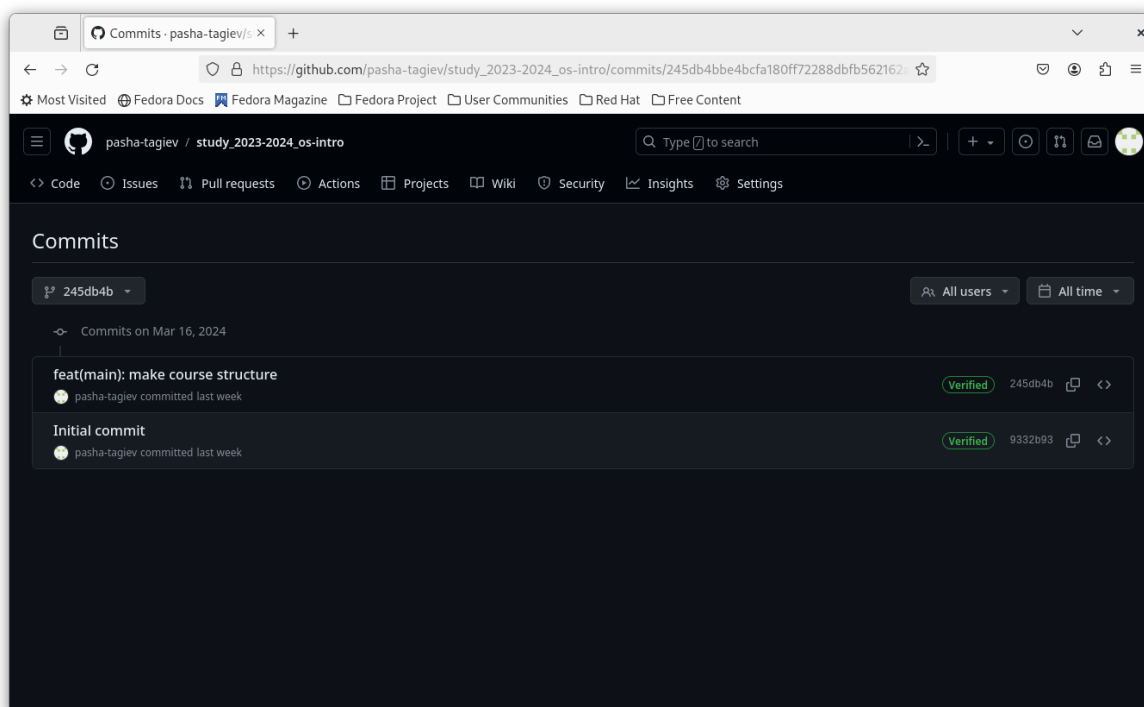


Рис. 4.16: Подписанные коммиты

## 5 Ответы на контрольные вопросы

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

Системы контроля версий (Version Control System, VCS) — это программные инструменты, помогающие командам разработчиков управлять изменениями в исходном коде с течением времени. В свете усложнения сред разработки они помогают командам разработчиков работать быстрее и эффективнее.

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

1. Репозиторий(хранилище) Git — это виртуальное хранилище проекта. В нем можно хранить версии кода для доступа по мере необходимости.

2. Commit — это команда в системе контроля версий Git, которая фиксирует изменения в репозитории.

3. История - список коммитов, который можно посмотреть командой `git log`.

4. Рабочая копия - рабочая копия является снимком(коммитом) одной из версий проекта.

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

- централизованные - история версий хранится на удалённом сервере,

а рабочий код на нескольких компьютерах. Компьютеры связаны с одним сервером. Примером может послужить SVN (Subversion).

- децентрализованные - рабочий код хранится на нескольких компьютерах, а история всех версий хранится как на удалённом сервере, так и на каждом из этих компьютеров. Все компьютеры связаны с сервером, но ещё дополнительно связаны между собой. Пример Git.

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

1. Инициализация репозитория
2. Создание рабочей копии
3. Внесение изменений
4. Коммит изменений
5. Просмотр истории
6. Обновление рабочей копии

5. Опишите порядок работы с общим хранилищем VCS.

1. Получение последней версии проекта
2. Внесение изменений
3. Фиксация изменений
4. Обновление рабочей копии
5. Разрешение конфликтов
6. Просмотр истории
7. Отправка изменений в общий репозиторий

6. Каковы основные задачи, решаемые инструментальным средством git?

- Хранить информацию и всех изменениях в коде, с возможностью в любой момент перейти к любому из них.
- Обеспечение удобства командной работы над кодом.

7. Назовите и дайте краткую характеристику командам git.

1. `git init` - создание основаного дерева репозитория.

2. `git pull` - получение изменений текущего дерева из центрального репозитория.
  3. `git push` - отправка изменений в центральный репозиторий.
  4. `git status` - просмотр измененных файлов.
  5. `git diff` - просмотр изменений.
  6. `git add` - добавить изменения.
  7. `git commit` - фиксация изменений.
8. Приведите примеры использования при работе с локальным и удалённым репозиториями.
- локальный репозиторий: `git commit -am "мой коммит"`.
  - удаленный репозиторий: `git push origin master` - отправка изменений на удаленный репозиторий origin, на ветку master.
9. Что такое и зачем могут быть нужны ветви (branches)?
- Ветка - параллельный участок истории в одном хранилище, между ветками возможно слияние. Обычно используются для создания новых функций или новых версий приложения.
10. Как и зачем можно игнорировать некоторые файлы при commit?
- Можно просто не добавлять их в индекс командой `git add` или создать файл `.gitignore`, в котором перечислить все файлы и папки которые требуется игнорировать. Может понадобится игнорировать настройки IDE, или бинарные файлы. Так как они зависят от конкретного разработчика и платформы, и в репозитории они могут быть лишними.

## 6 Выводы

В этой лабораторной работе мы научились настраивать git генерировать ключи для ssh и gpg, а также взаимодействовать с удаленными репозиториями, создавая свои из шаблонов и загружая изменения в локальном репозитории на удаленный.



## Список литературы

1. Что такое Git? [Электронный ресурс]. 2024. URL: <https://www.atlassian.com/ru/git/tutorials/what-is-git>.
2. Что такое подсистема Windows для Linux [Электронный ресурс]. 2023. URL: <https://learn.microsoft.com/ru-ru/windows/wsl/about>.