

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

## ОТЧЕТ

### ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

дисциплина:     *Архитектура компьютера*

Студент: Павличенко Родион Андреевич

Группа: НПИбд-02-24

МОСКВА

2025 г.

## Цель работы:

Цель работы: изучить идеологию и применение средств контроля версий.

Освоить умения по работе с git.

## Ход работы:

1) Переходим в режим суперпользователя, устанавливаем пакет git и пакет gh

```
rapavlichenko@rapavlichenko:~$ sudo -i
[sudo] пароль для rapavlichenko:
root@rapavlichenko:~# dnf install git
Updating and loading repositories:
Repositories loaded.
Пакет "git-2.47.0-1.fc41.x86_64" уже установлен.

Nothing to do.
root@rapavlichenko:~# dnf install gh
```

2) Задаем имя и email владельца репозитория (команды 1,2) , настроим utf-8 (команда 3), зададим имя начальной ветке (команда 4), введем параметры autocrlf и safecrlf (команды 5,6)

```
root@rapavlichenko:~# git config --global user.name "rapavlic
henko"
root@rapavlichenko:~# git config --global user.email "1132246
838@pfur.ru"
root@rapavlichenko:~# git config --global core.quotepath fals
e
root@rapavlichenko:~# git config --global init.defaultBranch
master
root@rapavlichenko:~# git config --global core.autocrlf input
root@rapavlichenko:~# git config --global core.safecrlf warn
root@rapavlichenko:~#
```

3) Создаем ключ ssh размером 4096 бит

```

root@rapavlichenko:~# ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:Ao8BTzGvNJd1+pThuBFnbKnKoT9BWKrTgaKJwwHd9+I root@rapav
lichenko
The key's randomart image is:
+---[RSA 4096]-----+
|  ...+.  o.=.  |
| . .+.000 0+o  |
| . .*=+.+o+   |
| .....+0+  o=  |
| +o.00*o+S .  |
| =.o o E.     |
|  . . . .    |
|    o        |
|  .          |
+---[SHA256]-----+

```

4) Создаем ключ по алгоритму ed25519

```

root@rapavlichenko:~# ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_ed25519
Your public key has been saved in /root/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:3PE9cnjN/Ts4UTuWA0VumwTtC8QDBwODtln/Q8Nz3QE root@rapav
lichenko
The key's randomart image is:
+---[ED25519 256]--+
|      .o.+o.E..+ |
|      . ...= +o .o|
|      . + o=,+o.  |
|      + ..+=+o+   |
|      S ..*B**=   |
|      ++Bo+       |
|      =. .        |
|                  |
+---[SHA256]-----+

```

5) Генерируем pgr ключ с определенными параметрами

```

root@rapavlichenko:~# gpg --full-generate-key
gpg (GnuPG) 2.4.5; 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 '/root/.gnupg'
Выберите тип ключа:
  (1) RSA and RSA
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
  (9) ECC (sign and encrypt) *default*
 (10) ECC (только для подписи)
 (14) Existing key from card
Ваш выбор? 1
длина ключей RSA может быть от 1024 до 4096.
Какой размер ключа Вам необходим? (3072) 4096
Запрошенный размер ключа - 4096 бит
Выберите срок действия ключа.
  0 = не ограничен
  <n> = срок действия ключа - n дней
  <n>w = срок действия ключа - n недель
  <n>m = срок действия ключа - n месяцев
  <n>y = срок действия ключа - n лет
Срок действия ключа? (0) 0
Срок действия ключа не ограничен
Все верно? (y/N) y


```

б) Копируем PGP-ключ в буфер обмена и переходим на github, вставляем полученный ключ

## GPG keys

New GPG key

This is a list of GPG keys associated with your account. Remove any keys that you do not recognize.



GPG

Lab

Email address: 1132246838@pfur.ru

Key ID: 13F4A6BD1AF9B58E

Subkeys: D82A472EFECB7607

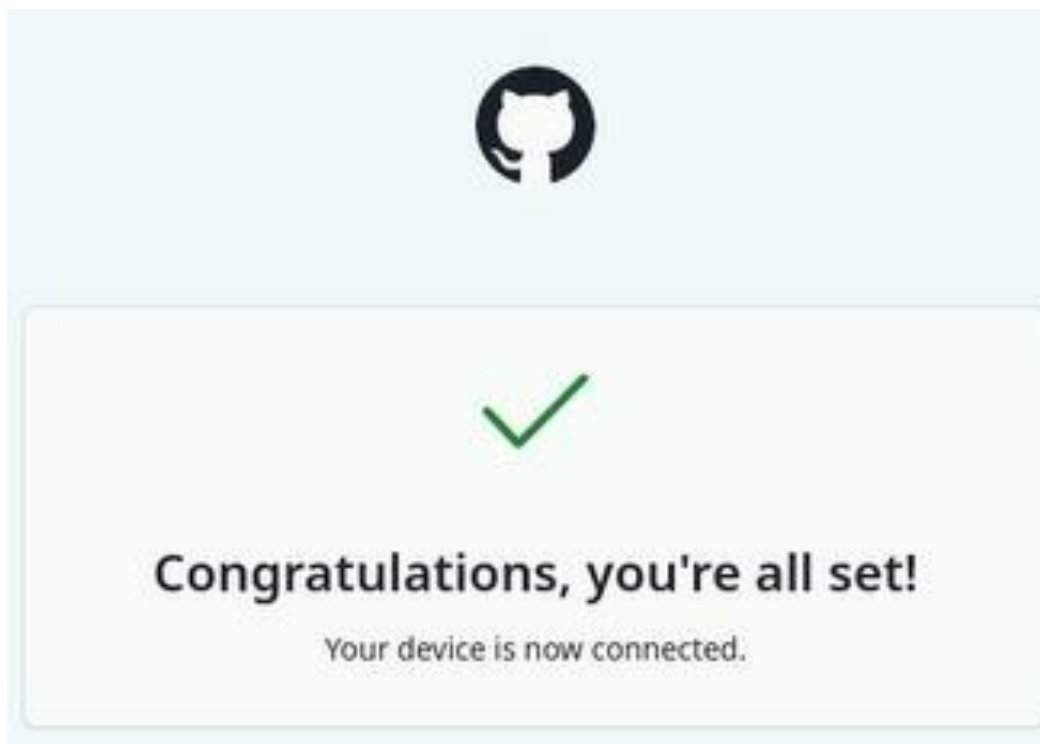
Added on Mar 1, 2025

Delete

7) Вводим email и указываем git применять его при подписи коммитов, вводим три команды

```
root@rapavlichenko:~# git config --global user.signingkey 1132246838@pfur.ru~
root@rapavlichenko:~# git config --global commit.gpgsign true
root@rapavlichenko:~# git config --global gpg.program $(which gpg2)
```

8) При помощи gh авторизируемся



9) Создаем репозиторий курса на основе шаблона

```
rapavlichenko@rapavlichenko:~$ mkdir -p ~/work/study/2022-2023/"Операционные системы"
rapavlichenko@rapavlichenko:~$ cd ~/work/study/2022-2023/"Операционные системы"
rapavlichenko@rapavlichenko:~/work/study/2022-2023/Операционные системы$
```

```
rapavlichenko@rapavlichenko:~/work/study/2022-2023/Операционные системы$ gh repo create study_2022-2023_os-intro --template=yamadharma/course-directory-student-template --public
```

10) Настраиваем каталог курса, сначала переходим в него (команда 1), далее удаляем лишние файлы (команда 2) и создаем необходимые каталоги (команда 3,4)

```

rapavlichenko@rapavlichenko:~/work/study/2022-2023/Операционн
ые системы$ cd ~/work/study/2022-2023/"Операционные системы"/
os-intro
rapavlichenko@rapavlichenko:~/work/study/2022-2023/Операционн
ые системы/os-intro$ rm package.json
rapavlichenko@rapavlichenko:~/work/study/2022-2023/Операционн
ые системы/os-intro$ echo os-intro > COURSE
rapavlichenko@rapavlichenko:~/work/study/2022-2023/Операционн
ые системы/os-intro$ make
Usage:
  make <target>

Targets:
  list                List of courses
  prepare             Generate directories struct
ure
  submodule I         Update submules

rapavlichenko@rapavlichenko:~/work/study/2022-2023/Операционн
ые системы/os-intro$ make prepare

```

11) Отправляем файлы на сервер

```

ые системы/os-intro$ git add .
rapavlichenko@rapavlichenko:~/work/study/2022-2023/Операционн
ые системы/os-intro$ git commit -am 'feat(main): make course
structure'

```

12) Ответы на контрольные вопросы:

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

Системы контроля версий (Version Control Systems, VCS) — это инструменты, позволяющие отслеживать изменения в коде, документации или любых других файлах, управлять разными версиями проекта и работать в команде без потери данных.

Основные задачи:

- Хранение истории изменений файлов
- Возможность отката к предыдущ
- Разрешение конфликтов при совместной работе
- Управление параллельной разработкой (ветвление и слияние)
- Обеспечение резервного копирования

2. *Объяснение понятий VCS и их отношений:*

Хранилище (репозиторий) — это база данных, содержащая все файлы проекта и историю их изменений. Может быть локальным или удалённым.

Commit — фиксация изменений в репозитории. Каждый коммит содержит снимок



изменённых файлов и метаданные (автор, дата, комментарий).

История — последовательность коммитов, отображающая все изменения в проекте с момента его создания.

Рабочая копия — текущая версия проекта на локальном компьютере разработчика, которая может содержать несохранённые изменения.

### *3. Централизованные и децентрализованные VCS:*

Централизованные VCS (CVCS) используют единый центральный сервер для хранения всех версий файлов. Разработчики работают с рабочими копиями и получают обновления с сервера.

Примеры: SVN (Subversion), Perforce, CVS

Децентрализованные VCS (DVCS) — каждый разработчик имеет полную копию репозитория, включая всю историю изменений. Работа возможна без подключения к серверу.

Примеры: Git, Mercurial, Fossil

### *4. Действия при единоличной работе с хранилищем (Git):*

Создание репозитория: `git init`

Добавление файлов: `git add <файл>`

Фиксация изменений: `git commit -m "Описание изменений"`

Просмотр истории: `git log`

Откат к предыдущей версии: `git checkout <commit_hash>`

Создание веток и переключение между ними: `git branch / git checkout`

### *5. Порядок работы с общим хранилищем (Git + удалённый репозиторий):*

Клонирование репозитория: `git clone <URL>`

Создание новой ветки: `git checkout -b feature_branch`

Добавление изменений: `git add .`

Фиксация изменений: `git commit -m "Описание изменений"`

Отправка изменений в удалённый репозиторий: `git push origin feature_branch`

Обновление локальной версии: `git pull`

Слияние изменений: `git merge feature_branch`

### *6. Основные задачи, решаемые Git:*

Отслеживание изменений в файлах

Ведение параллельных разработок с помощью

Слияние и разрешение конфликтов

Работа с локальными и удалёнными репозиториями

Откат к предыдущим версиям проекта

*7. Основные команды Git и их краткая характеристика:*

git init – создание нового локального репозитория

git clone <URL> – клонирование удалённого репозитория

git add <файл> – добавление файлов в индекс для следующего commit

git commit -m "сообщение" – создание commit'a

git status – проверка состояния репозитория

git log – просмотр истории commit'ов

git branch – просмотр и создание веток

git checkout <ветка> – переключение на другую ветку

git merge <ветка> – слияние веток

git push – отправка изменений в удалённый репозиторий

git pull – получение изменений из удалённого репозитория

git rebase – переписывание истории commit'

git stash – временное сохранение изменений без commit

*8. Примеры использования Git с локальными и удалёнными репозиториями:*

Локальная работа:

git init

git add .

git commit -m "Первый коммит"

Работа с удалённым репозиторием:

git clone https://github.com/user/repo.git

git pull origin main

git push origin main

*9. Что такое ветви (branches) и зач*

Ветви позволяют разрабатывать новые функции и исправлять ошибки параллельно, не изменяя основную версию кода. После завершения работы изменения объединяются с основной веткой (обычно main или master).

Пример создания и объединения ветки:



```
git checkout -b new-feature
```

# работа с кодом...

```
git commit -am "Добавлена новая фича"
```

```
git checkout main
```

```
git merge new-feature
```

```
git branch -d new-feature
```

## 10. *Как и зачем игнорировать файлы при commit?*

Некоторые файлы (например, логи, кэш, временные файлы, конфиденциальные данные) не должны попадать в репозиторий. Для их игнорирования используется файл .gitignore.

Пример .gitignore:

```
*.log
```

```
node_modules/
```

```
.env
```

Файл .gitignore предотвращает случайное добавление нежелательных файлов в репозиторий.

**Вывод:** мы изучили идеологию и применение средств контроля версий.

Освоили умения по работе с git.