

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2 _____

дисциплина: *Операционные системы*

Студент: Ниemek Яи Жак

Группа: НММБд-04-24

МОСКВА

2025__ г.

Цель работы

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

```
jacques@vbox:~$ sudo dnf install -y git gh gpg xclip
[sudo] Mot de passe de jacques :
Dernière vérification de l'expiration des métadonnées effectuée il y a 3:03:57 le mer. 27 août 2025 14:50:51.
Le paquet git-2.41.0-2.fc39.x86_64 est déjà installé.
Le paquet gnupg2-2.4.3-2.fc39.x86_64 est déjà installé.
Dépendances résolues.
=====
Paquet      Architecture  Version                Dépôt      Taille
=====
Installation:
gh           x86_64        2.45.0-1.fc39          updates    9.1 M
xclip       x86_64        0.13-20.git11cba61.fc39 fedora      37 k
=====
Résumé de la transaction
=====
Installer 2 Paquets

Taille totale des téléchargements : 9.2 M
Taille des paquets installés : 47 M
Téléchargement des paquets :
[MIRROR] xclip-0.13-20.git11cba61.fc39.x86_64.rpm: Curl error (7): Couldn't connect to server for https://fedora-archive.ip-connect.info/fedora/linux/releases/39/Everything/x86_64/os/Packages/x/xclip-0.13-20.git11cba61.fc39.x86_64.rpm [Failed to connect to fedora-archive.ip-connect.info port 443 after 26376 ms: Couldn't connect to server]
[MIRROR] gh-2.45.0-1.fc39.x86_64.rpm: Curl error (28): Timeout was reached for http://fedora-archive.ip-connect.vn.ua/fedora/linux/updates/39/Everything/x86_64/Packages/g/gh-2.45.0-1.fc39.x86_64.rpm [Failed to connect to fedora-archive.ip-connect.vn.ua port 80 after 30000 ms: Timeout was reached]
[MIRROR] xclip-0.13-20.git11cba61.fc39.x86_64.rpm: Curl error (7): Couldn't connect to server for http://fedora-archive.ip-connect.info/fedora/linux/releases/39/Everything/x86_64/os/Packages/x/xclip-0.13-20.git11cba61.fc39.x86_64.rpm [Failed to connect to fedora-archive.ip-connect.info port 80 after 21012 ms: Couldn't connect to server]
(1/2): xclip-0.13-20.git11cba61.fc39.x86_64.rpm 784 B/s | 37 kB 00:47
(2/2): gh-2.45.0-1.fc39.x86_64.rpm 182 kB/s | 9.1 MB 00:51
-----
Total 178 kB/s | 9.2 MB 00:52
Test de la transaction
La vérification de la transaction a réussi.
Lancement de la transaction de test
```

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`:

```
git config --global quotePath false
```

```
Terminé !
jacques@vbox:~$ git config --global user.name "Jacques"
jacques@vbox:~$ git config --global user.email "nyemeckyaijacques@gmail.com"
jacques@vbox:~$ git config --global core.quotePath false
jacques@vbox:~$ git config --global init.defaultBranch master
jacques@vbox:~$ git config --global core.autocrlf input
jacques@vbox:~$ git config --global core.safecrlf warn
jacques@vbox:~$
```

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

- Установим *git*:
- `dnf install git`

Установка gh

- Fedora:
- `dnf install gh`

Базовая настройка git

- Зададим 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`
- Настройте верификацию и подписание коммитов git (см. [Верификация коммитов git с помощью GPG](#)).
- Зададим 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`

Создайте ключи *ssh*

- по алгоритму *rsa* с ключём размером 4096 бит:
- `ssh-keygen -t rsa -b 4096`
- по алгоритму *ed25519*:
- `ssh-keygen -t ed25519`

Создайте ключи *pgp*

- Генерируем ключ
- `gpg --full-generate-key`
- Из предложенных опций выбираем:
 - тип *RSA and RSA*;
 - размер 4096;
 - выберите срок действия; значение по умолчанию — 0 (срок действия не истекает никогда).
- GPG запросит личную информацию, которая сохранится в ключе:
 - Имя (не менее 5 символов).
 - Адрес электронной почты.
 - При вводе email убедитесь, что он соответствует адресу, используемому на GitHub.
 - Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить это поле пустым.

Настройка github

- Создайте учётную запись на <https://github.com>.
- Заполните основные данные на <https://github.com>.

Добавление PGP ключа в GitHub

- Выводим список ключей и копируем 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			
- Скопируйте ваш сгенерированный PGP ключ в буфер обмена:
- `gpg --armor --export <PGP Fingerprint> | xclip -sel clip`
- Перейдите в настройки GitHub (<https://github.com/settings/keys>), нажмите на кнопку *New GPG key* и вставьте полученный ключ в поле ввода.

Настройка автоматических подписей коммитов git

- Используя введённый email, укажите 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 задаст несколько наводящих вопросов.
- Авторизоваться можно 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

- Необходимо 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`

```

jacques@vbox:~$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/jacques/.ssh/id_ed25519): key
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in key
Your public key has been saved in key.pub
The key fingerprint is:
SHA256:cw01G0b6TyWqLNbdQR3B5ja/GyoW3PBMEtzQKHK3quI jacques@vbox
The key's randomart image is:
+--[ED25519 256]--+
|      .o@ .++|
|      . o O *.o.|
|      o = + +..|
|      X + B.|
|      S B % + +|
|      * + * .|
|      . . . ...|
|      . . o ...|
|      .E. . . .|
+-----[SHA256]-----+
jacques@vbox:~$

```

Add new SSH Key

Title

key jacques

Key type

Authentication Key ▾

Key

SHA256:cw01G0b6TyWqINbdQR3B5ja/GyoW3PBMEtzQKHK3qul

```
+-----[SHA256]-----+
jacques@vbox:~$ gpg --full-generate-key
gpg (GnuPG) 2.4.3; Copyright (C) 2023 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: répertoire « /home/jacques/.gnupg » créé
Sélectionnez le type de clef désiré :
  (1) RSA and RSA
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
  (9) ECC (sign and encrypt) *default*
 (10) ECC (signature seule)
 (14) Existing key from card
Quel est votre choix ? 1
les clefs RSA peuvent faire une taille comprise entre 1024 et 4096 bits.
Quelle taille de clef désirez-vous ? (3072) 4096
La taille demandée est 4096 bits
Veuillez indiquer le temps pendant lequel cette clef devrait être valable.
  0 = la clef n'expire pas
  <n> = la clef expire dans n jours
  <n>w = la clef expire dans n semaines
  <n>m = la clef expire dans n mois
  <n>y = la clef expire dans n ans
Pendant combien de temps la clef est-elle valable ? (0) 0
La clef n'expire pas du tout
Est-ce correct ? (o/N) o

GnuPG doit construire une identité pour identifier la clef.

Nom réel : jacques+nyameckyaijacques@gmail.com
Adresse électronique : nyameckyaijacques@gmail.com
Commentaire : gpg key
Vous avez sélectionné cette identité :
  « jacques+nyameckyaijacques@gmail.com (gpg key) <nyameckyaijacques@gmail.com> »
```

```

GnuPG doit construire une identité pour identifier la clef.

Nom réel : jacques+nyemeckyaijacques@gmail.com
Adresse électronique : nyemeckyaijacques@gmail.com
Commentaire : gpg key
Vous avez sélectionné cette identité :
  « jacques+nyemeckyaijacques@gmail.com (gpg key) <nyemeckyaijacques@gmail.com> »

Changer le (N)om, le (C)ommentaire, l'(A)dresse électronique
ou (O)ui/(Q)uitter ? N
Nom réel : jacques
Vous avez sélectionné cette identité :
  « jacques (gpg key) <nyemeckyaijacques@gmail.com> »

Changer le (N)om, le (C)ommentaire, l'(A)dresse électronique
ou (O)ui/(Q)uitter ? O
De nombreux octets aléatoires doivent être générés. Vous devriez faire
autre chose (taper au clavier, déplacer la souris, utiliser les disques)
pendant la génération de nombres premiers ; cela donne au générateur de
nombres aléatoires une meilleure chance d'obtenir suffisamment d'entropie.
De nombreux octets aléatoires doivent être générés. Vous devriez faire
autre chose (taper au clavier, déplacer la souris, utiliser les disques)
pendant la génération de nombres premiers ; cela donne au générateur de
nombres aléatoires une meilleure chance d'obtenir suffisamment d'entropie.
gpg: /home/jacques/.gnupg/trustdb.gpg : base de confiance créée
gpg: répertoire « /home/jacques/.gnupg/openpgp-revocs.d » créé
gpg: revocation certificate stored as '/home/jacques/.gnupg/openpgp-revocs.d/69467C81DE786D0E7CE4449AED92BC4B4568035B.rev'
les clefs publique et secrète ont été créées et signées.

pub  rsa4096 2025-08-27 [SC]
    69467C81DE786D0E7CE4449AED92BC4B4568035B
uid                          jacques (gpg key) <nyemeckyaijacques@gmail.com>
sub  rsa4096 2025-08-27 [E]

jacques@vbox:~$

```

```

jacques@vbox:~$ gpg --list-secret-keys --keyid-format LONG
gpg: vérification de la base de confiance
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: profondeur : 0  valables : 1  signées : 0
   confiance : 0 i., 0 n.d., 0 j., 0 m., 0 t., 1 u.
[keyboard]
-----
sec  rsa4096/ED92BC4B4568035B 2025-08-27 [SC]
    69467C81DE786D0E7CE4449AED92BC4B4568035B
uid                          [ ultime ] jacques (gpg key) <nyemeckyaijacques@gmail.com>
ssb  rsa4096/6F1786E2C88CD5FD 2025-08-27 [E]

```

```

jacques@vbox:~$ gpg --armor --export <PGP_Fingerprint>
bash: erreur de syntaxe près du symbole inattendu « newline »
jacques@vbox:~$ gpg --armor --export <PGP_Fingerprint> | xclip -sel clip
bash: erreur de syntaxe près du symbole inattendu « | »
jacques@vbox:~$ git config --global user.signingkey <PGP_Fingerprint>
bash: erreur de syntaxe près du symbole inattendu « newline »
jacques@vbox:~$ git config --global commit.gpgsign true
jacques@vbox:~$ git config --global gpgpg.program $(which gpg2)
jacques@vbox:~$

```

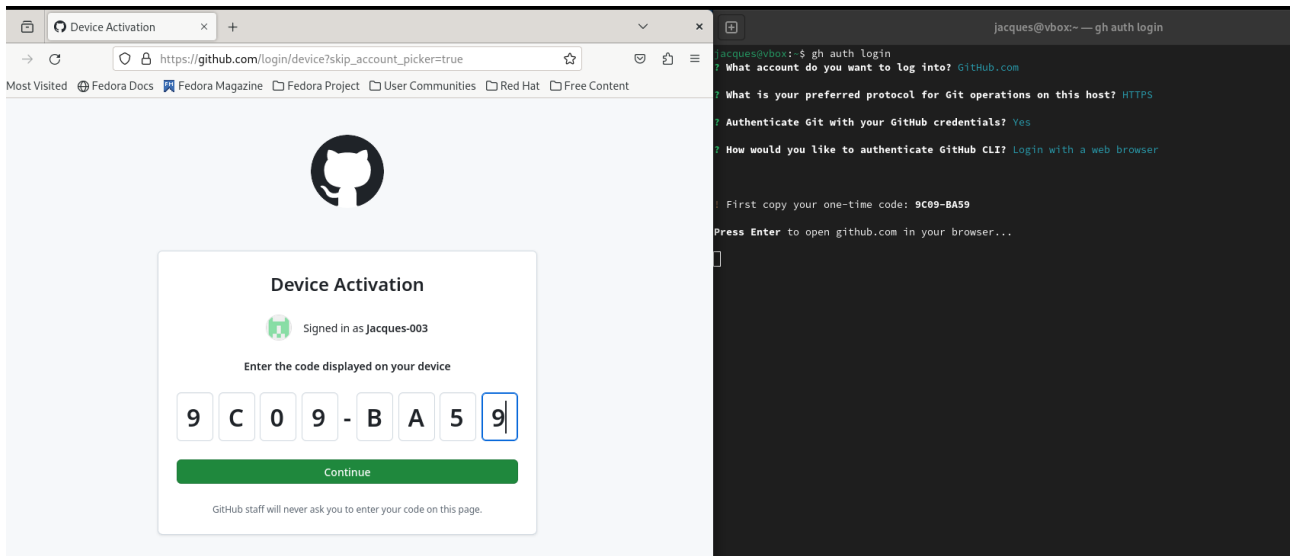
Vigilant mode

✓ 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.

[Learn about vigilant mode.](#)



```
✓ Authentication complete.
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as Jacques-003
jacques@vbox:~$ mkdir -p ~/work/study/2024-2025/"Операционные системы"
jacques@vbox:~$ cd ~/work/study/2024-2025/"Операционные системы"
jacques@vbox:~/work/study/2024-2025/Операционные системы$ gh repo create study_2024-2025-os-intro \ --template=yamadha
rma/course-directory-student-template \ --public
accepts at most 1 arg(s), received 3
jacques@vbox:~/work/study/2024-2025/Операционные системы$ git clone --recursive git@hithub.com:<Jacques-003>/study_20
24-2025-os-intro.git os-intro
bash: Jacques-003: Aucun fichier ou dossier de ce type
jacques@vbox:~/work/study/2024-2025/Операционные системы$ cd ~/work/study/2024-2025/"Операционные системы"/os-intro
bash: cd: /home/jacques/work/study/2024-2025/Операционные системы/os-intro: Aucun fichier ou dossier de ce type
jacques@vbox:~/work/study/2024-2025/Операционные системы$ rm package.json
rm: impossible de supprimer 'package.json': Aucun fichier ou dossier de ce type
jacques@vbox:~/work/study/2024-2025/Операционные системы$
```

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

Системы контроля версий (VCS) — это инструменты, которые позволяют хранить и управлять изменениями файлов и проектов. Основные задачи:

- * сохранение истории изменений;
- * отслеживание и сравнение версий;
- * совместная работа нескольких разработчиков;
- * возможность отката к предыдущим состояниям.

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

- * Хранилище (репозиторий) — база данных, в которой хранятся все версии файлов и история изменений.
- * Commit — фиксирование изменений в хранилище.

* История — последовательность commit-ов с информацией о времени, пользователе и содержании изменений.

* Рабочая копия — локальные файлы проекта, с которыми работает пользователь.

С в я з ь : р а б о ч а я к о п и я → и з м е н е н и я → c o m m i t →
с о х р а н я е т с я в х р а н и л и щ е и с т а н о в и т с я ч а
с т ь ю и с т о р и и .

3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Примеры.

* Централизованные VCS (CVCS): один центральный сервер, все разработчики получают и отправляют изменения туда. Пример: *Subversion (SVN)*.

* Децентрализованные VCS (DVCS): каждый разработчик имеет полную копию репозитория, обмен идёт через синхронизацию. Примеры: *Git, Mercurial*.

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

1. Создание репозитория.
2. Добавление файлов.
3. Выполнение commit-ов для сохранения изменений.
4. Просмотр истории и работа с версиями.

5. Опишите порядок работы с общим хранилищем VCS.

1. Клонирование удалённого репозитория.
2. Создание ветви или внесение изменений в рабочую копию.
3. Commit изменений.
4. Отправка изменений на сервер (`push`).
5. Получение изменений от других (`pull`, `fetch`).
6. Решение конфликтов при необходимости.

6. Каковы основные задачи, решаемые инструментальным средством Git?

- * Отслеживание изменений и версий файлов.
- * Совместная работа над проектами.
- * Создание веток для параллельной разработки.
- * Слияние изменений.
- * Работа как с локальными, так и с удалёнными репозиториями.

7. Назовите и дайте краткую характеристику командам git.

- * `git init` — создание репозитория.
- * `git clone` — клонирование репозитория.
- * `git status` — проверка состояния файлов.
- * `git add` — добавление файлов в индекс.
- * `git commit -m "msg"` — сохранение изменений.
- * `git log` — просмотр истории.
- * `git branch` — управление ветками.
- * `git checkout` — переключение ветвей.
- * `git merge` — слияние веток.
- * `git push` — отправка изменений на сервер.
- * `git pull` — получение и объединение изменений.

8. Примеры использования при работе с локальным и удалённым репозиториями.

- * Локально:

```
git init
git add file.txt
git commit -m "Добавлен новый файл"
```

- * С удалённым репозиторием:

```
git clone https://github.com/user/repo.git
git push origin main
git pull origin main
```

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

Ветви — это независимые линии разработки в проекте. Они нужны для:

- * параллельной работы над новыми функциями;
- * тестирования без риска сломать основную версию;
- * удобного объединения разных направлений разработки.

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

Файлы игнорируются через `.gitignore`. Это нужно, чтобы не добавлять в репозиторий:

- * временные файлы;
- * логи;
- * локальные настройки IDE;
- * скомпилированные бинарные файлы.

Пример `.gitignore`:

*.log
*.tmp
node_modules/
venv/