**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ**

**по лабораторной работе № 2**

*дисциплина: Архитектура компьютера*

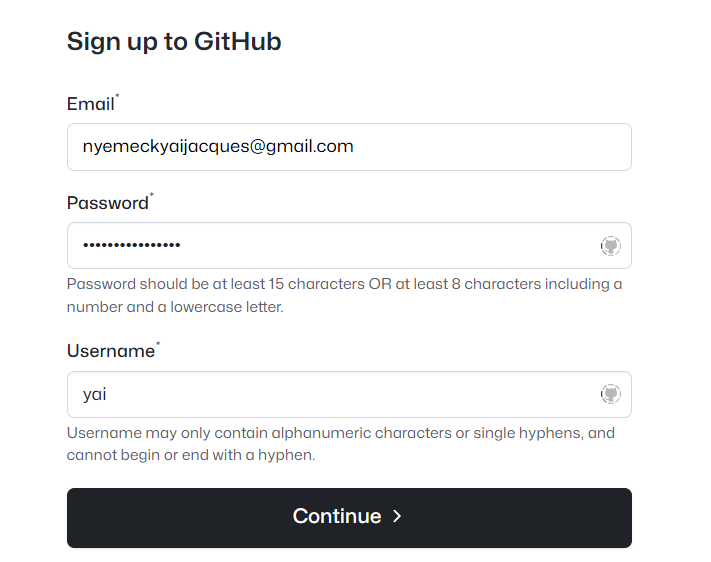
Студент: Ниемек Яи Жак

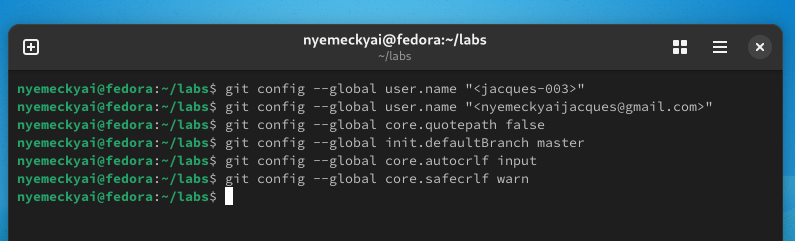
Группа: НММБд-04-24

**МОСКВА**

2025 г.

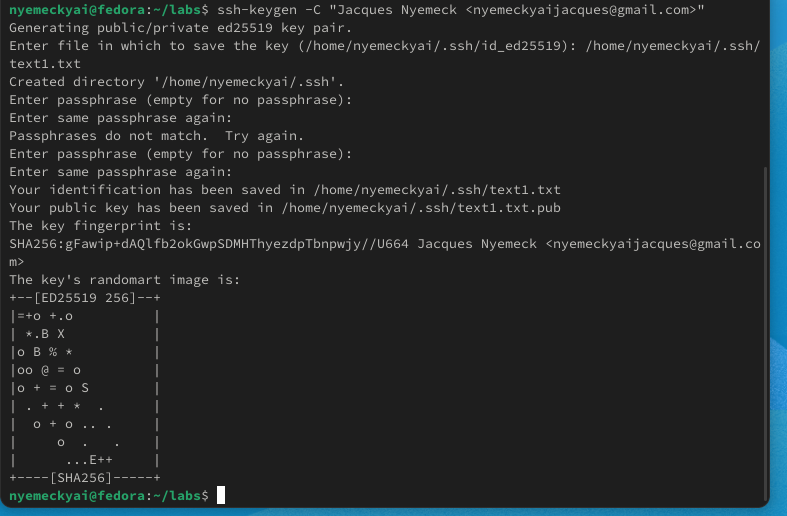
**2.4.2. Базовая настройка Git**

1. **Настройка имени и email**
   * Git использует эти данные для подписи коммитов.
   * Вводим:
   * git config --global user.name "<Name Surname>"
   * git config --global user.email "<work@mail>"
   * <Name Surname> и <work@mail> замените на свои данные.
2. **Настройка кодировки UTF-8**
   * Чтобы Git корректно отображал русские символы в сообщениях:
   * git config --global core.quotepath false
3. **Настройка имени главной ветки**
   * По умолчанию в Git основная ветка называется master:
   * git config --global init.defaultBranch master
4. **Настройка обработки переноса строк**
   * autocrlf помогает избегать проблем с переносами строк между Windows и Linux/Mac:
   * git config --global core.autocrlf input
   * safecrlf предупреждает, если файлы содержат смешанные типы переноса строк:
   * git config --global core.safecrlf warn



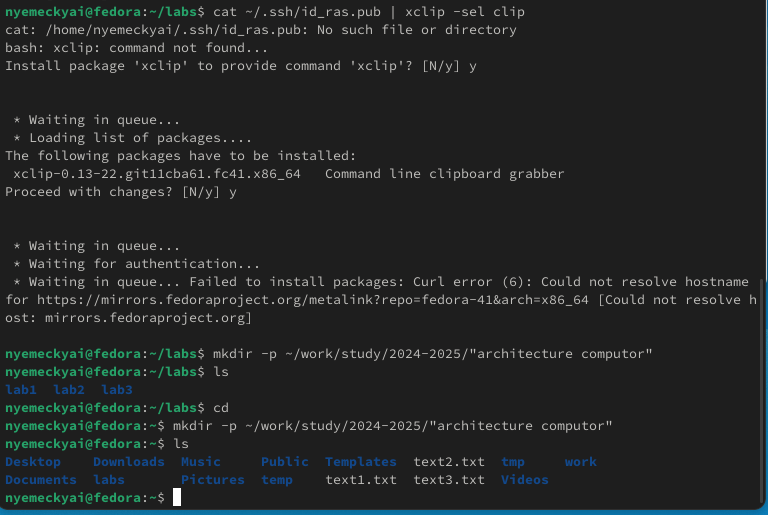
**2.4.3. Создание SSH-ключа**

1. **Генерация SSH-ключа**
   * SSH-ключи позволяют безопасно работать с удалёнными репозиториями без ввода пароля.
   * Вводим:
   * ssh-keygen -C "Имя Фамилия <work@mail>"
   * Ключи сохраняются в папке ~/.ssh/ (домашний каталог).
2. **Добавление SSH-ключа в GitHub**
   * Открываем GitHub → Settings → SSH and GPG keys.
   * Нажимаем New SSH key, вставляем ключ, задаём название.
   * Копируем ключ:
   * cat ~/.ssh/id\_rsa.pub | xclip -sel clip
   * Вставляем в GitHub.

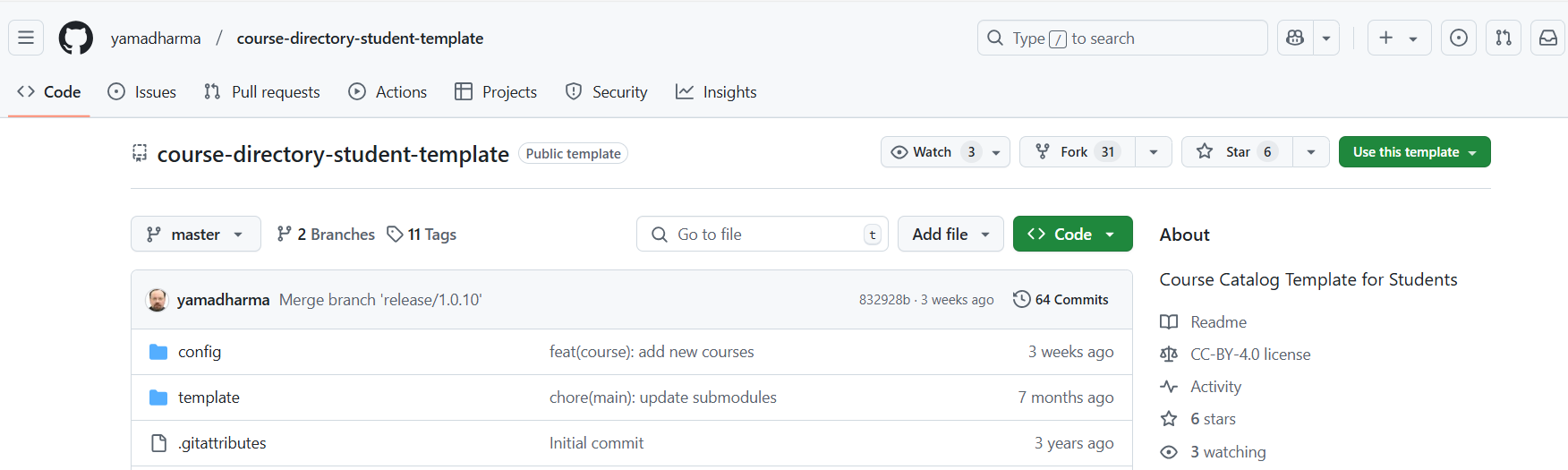


**2.4.4. Создание рабочего пространства**

1. **Создаём нужные папки**
   * Организуем файлы для удобства работы:
   * mkdir -p ~/work/study/2023-2024/"Архитектура компьютера"
   * Теперь у нас есть папка Архитектура компьютера внутри 2023-2024.

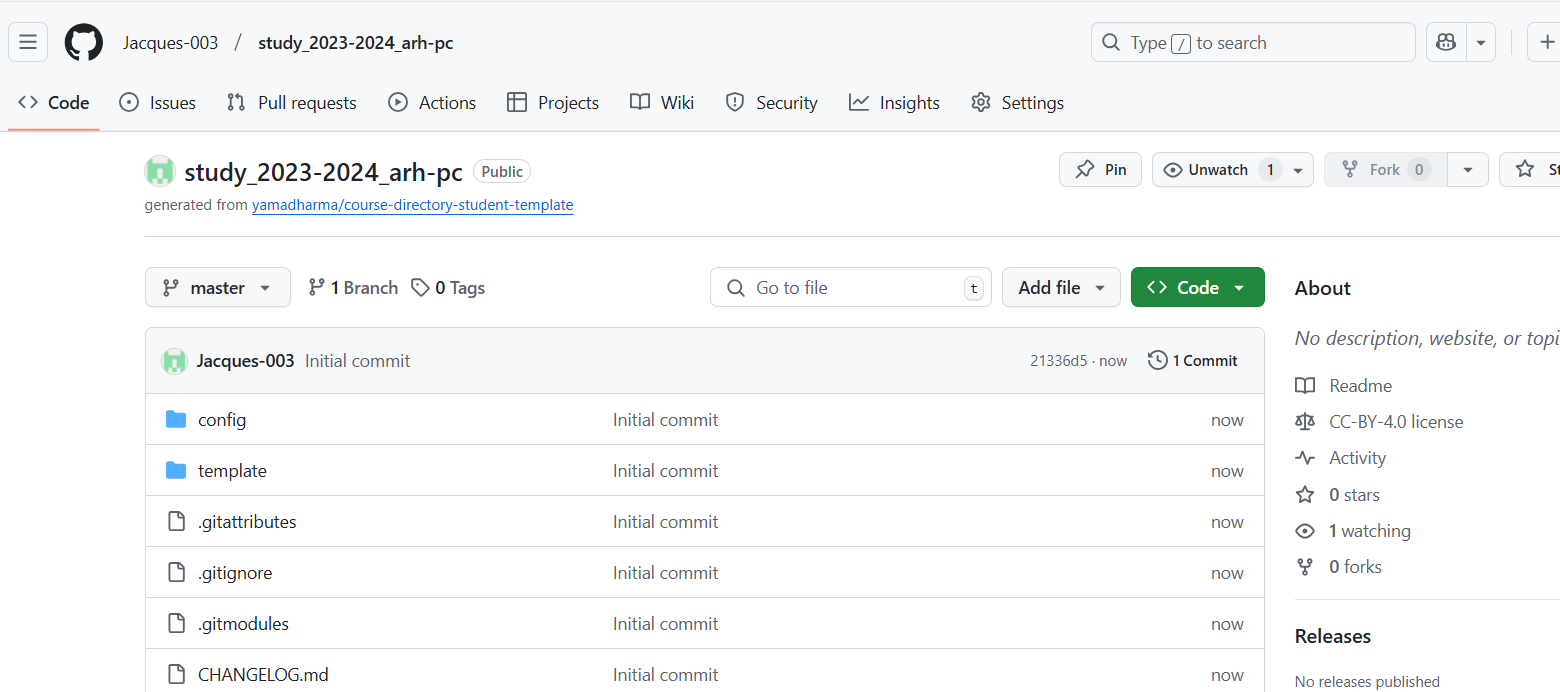


**2.4.5. Создание репозитория на основе шаблона**

1. **Создание репозитория через GitHub**
   * Открываем [шаблон](https://github.com/yamadharma/course-directory-student-template).
   * Нажимаем Use this template.
   * Вводим название: study\_2023-2024\_arch-pc.
   * Создаём репозиторий.
2. **Клонирование репозитория**
   * Переходим в созданную папку:
   * cd ~/work/study/2023-2024/"Архитектура компьютера"
   * Копируем SSH-ссылку из GitHub.
   * Клонируем репозиторий:
   * git clone --recursive git@github.com:<user\_name>/study\_2023-2024\_arch-pc.git arch-pc
   * Теперь папка arch-pc содержит файлы из репозитория.

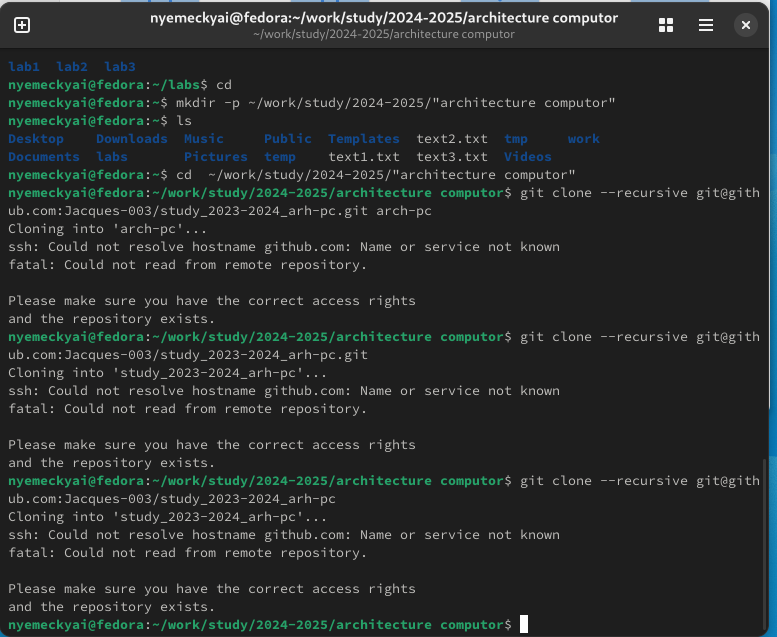
**2.4.6. Настройка каталога курса**

1. **Переход в каталог проекта**
2. cd ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc
3. **Удаление ненужных файлов**
4. rm package.json
5. **Создание файлов для курса**
   * Создаём файл с названием курса:
   * echo arch-pc > COURSE
   * Запускаем команду make (если используется Makefile).
6. **Отправка изменений на GitHub**
7. git add .
8. git commit -am 'feat(main): make course structure'
9. git push
   * git add . – добавляет все файлы в коммит.
   * git commit -am '...' – сохраняет изменения с сообщением.
   * git push – загружает их на сервер.



**2.5. Задание для самостоятельной работы**

1. **Создание отчёта по лабораторной**
   * Создаём папку:
   * mkdir -p ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab02/report
   * Пишем отчёт в файле (например, report.txt).
2. **Копирование старых отчётов**
   * Если у вас есть отчёты по предыдущим лабораторным, копируем их в нужные каталоги.
   * Например:
   * cp ~/старый\_каталог/lab01/report.txt ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab01/
3. **Загрузка файлов на GitHub**
4. git add .
5. git commit -am 'feat(lab02): add lab report'
6. git push
   * После этого можно проверить файлы на GitHub.



ответы на контрольные вопросы по Git и системам контроля версий:

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

Системы контроля версий (Version Control System, VCS) — это инструменты, которые отслеживают изменения в файлах и позволяют управлять их разными версиями. Они предназначены для:

* Хранения истории изменений
* Возможности отката к предыдущим версиям
* Организации работы нескольких пользователей над одним проектом
* Обнаружения и разрешения конфликтов при изменении файлов

**2. Понятия VCS и их взаимоотношения: хранилище, commit, история, рабочая копия**

* **Хранилище (репозиторий, repository)** — место, где хранятся все версии файлов и их изменения.
* **Commit** — фиксированное изменение в репозитории. Каждый коммит содержит описание изменений и создаёт новую версию.
* **История (history)** — последовательность коммитов, показывающая, как изменялся проект с течением времени.
* **Рабочая копия (working copy)** — версия файлов, с которой работает пользователь, загруженная из репозитория.

**3. Централизованные и децентрализованные VCS, примеры**

* **Централизованные VCS (CVCS)** — один общий сервер хранит всю историю изменений, а пользователи работают с копиями файлов. Пример: **Subversion (SVN), Perforce**.
* **Децентрализованные VCS (DVCS)** — каждый разработчик имеет полную копию репозитория. Пример: **Git, Mercurial**.

**4. Действия при работе с VCS в одиночку**

1. Создание репозитория (git init)
2. Добавление файлов под контроль версии (git add)
3. Фиксация изменений (git commit -m "Сообщение")
4. Просмотр истории (git log)
5. Восстановление старой версии при необходимости (git checkout)

**5. Порядок работы с общим хранилищем VCS**

1. **Клонирование репозитория** (git clone)
2. **Создание новой ветки** (git checkout -b feature\_branch)
3. **Работа с файлами и коммиты** (git add, git commit)
4. **Обновление локальной версии** (git pull)
5. **Разрешение конфликтов при слиянии**
6. **Отправка изменений на сервер** (git push)
7. **Создание pull request (если используется GitHub/GitLab)**

**6. Основные задачи, решаемые Git**

* Управление версиями кода
* Возможность работать в команде
* Создание веток для отдельных задач
* Откат изменений и работа с историей коммитов
* Удобное разветвление и слияние кода

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

* git init — создать новый репозиторий
* git clone <url> — скопировать удалённый репозиторий
* git add <файл> — добавить файлы в коммит
* git commit -m "Описание" — зафиксировать изменения
* git status — проверить статус файлов
* git log — посмотреть историю коммитов
* git branch — список веток
* git checkout <ветка> — переключиться на ветку
* git merge <ветка> — слить изменения из другой ветки
* git pull — скачать обновления с сервера
* git push — отправить изменения на сервер

**8. Примеры работы с локальным и удалённым репозиториями**

**Локальный репозиторий**

git init

echo "Hello, Git" > file.txt

git add file.txt

git commit -m "Первый коммит"

**Работа с удалённым репозиторием**

git remote add origin git@github.com:user/project.git

git push -u origin master

git pull origin master