

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: Гаджиев Мирзе Керимович

Группа: НКАбд-05-24

МОСКВА

2024 г.

Содержание

1 Цель работы.....	3
2 Задание.....	4
3 Теоретическое введение.....	5
4 Выполнение лабораторной работы.....	6
4.1 Создание учетной записи GitHub.....	6
4.2 Базовая настройка Git.....	6
4.3 Создание SSH-ключа.....	6
4.4 Создание рабочего пространства и репозитория курса на основе шаблона..	7
4.5 Настройка каталога курса.....	8
5 Выполнение заданий для самостоятельной работы.....	9
6 Выводы.....	10
7 Вопросы для самопроверки.....	10

1 Цель работы

Целью работы является изучить идеологию и применение средств контроля версий. Приобрести практические навыки по работе с системой git.

2 Задание

1. Настройка GitHub.
2. Базовая настройка Git.
3. Создание SSH-ключа.
4. Создание рабочего пространства и репозитория курса на основе шаблона.
5. Создание репозитория курса на основе шаблона.
6. Настройка каталога курса.
7. Выполнение заданий для самостоятельной работы.

3 Теоретическое введение

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. Архитектура ЭВМ 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.

4 Выполнение лабораторной работы

4.1 Создание учетной записи GitHub

Я создал учетную запись на GitHub, заполнил основные данные.

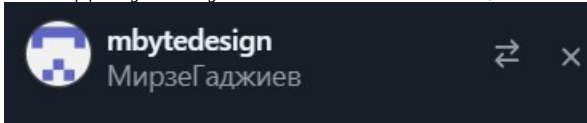


Рис.1. Учётная запись

4.2 Базовая настройка Git

Открываю виртуальную машину. Открываю терминал и делаю предварительную конфигурацию git. Ввожу команду `git config --global user.name ""`, Указываю свое имя и команду `git config --global user.email "work@mail"`, указывая в ней мою электронную почту.

```
liveuser@localhost-live:~$ git config --global user.name "<Gadzhiev Mirze>"
liveuser@localhost-live:~$ git config --global user.email "<1032240486@pfur.ru>"
```

Рис. 2. Базовая настройка Git

Настраиваю utf-8 в выводе сообщений git.

```
liveuser@localhost-live:~$ git config --global core.quotePath false
```

Рис. 3. Настройка кодировки

Задаю имя начальной ветки, параметр `autocrlf` со значением `input`, чтобы конвертировать CRLF в L, параметр `safecrlf` со значением `warn`.

```
liveuser@localhost-live:~$ git config --global init.defaultBranch master
liveuser@localhost-live:~$ git config --global core.autocrlf input
liveuser@localhost-live:~$ git config --global core.safecrlf warn
```

Рис. 4. Введение параметров

4.3 Создание SSH-ключа

Для последующей идентификации пользователя на сервере репозитория необходимо сгенерировать пару ключей (приватный и открытый). Ввожу команду `ssh-keygen -C "Имя Фамилия, work@email"`, указывая имя владельца и электронную почту владельца. Ключ автоматически сохранится в каталоге `~/.ssh/`.

```
liveuser@localhost-live:~$ ssh-keygen -C "Gadzhiev Mirze <1032240486@pfur.ru>"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/liveuser/.ssh/id_ed25519):
Created directory '/home/liveuser/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/liveuser/.ssh/id_ed25519
Your public key has been saved in /home/liveuser/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:KYEbaU0ZQ0vxaXEBwE1f8ZpBB2rdtIQhmjhQid5+tTU Gadzhiev Mirze <1032240486@pfur.ru>
The key's randomart image is:
+--[ED25519 256]--+
| .==*++o *==+ |
| ..o++* B.* . |
| . * X * o + |
| o * O o E |
| o = o S . |
| o . o |
| . |
+-----[SHA256]-----+
```

Рис. 5. Генерация ключа

Открываю браузер, захожу на сайт GitHub. Открываю свой профиль и выбираю страницу «SSH and GPG keys». Нажимаю кнопку «New SSH key». Вставляю скопированный ключ в

поле «Key». В поле Title указываю имя для ключа. Нажимаю «Add SSH-key», чтобы завершить добавление ключа.

```
liveuser@localhost-live:~$ cat ~/.ssh/id_ed25519.pub | xclip -sel clip
```

Рис. 6. Копирование ключа

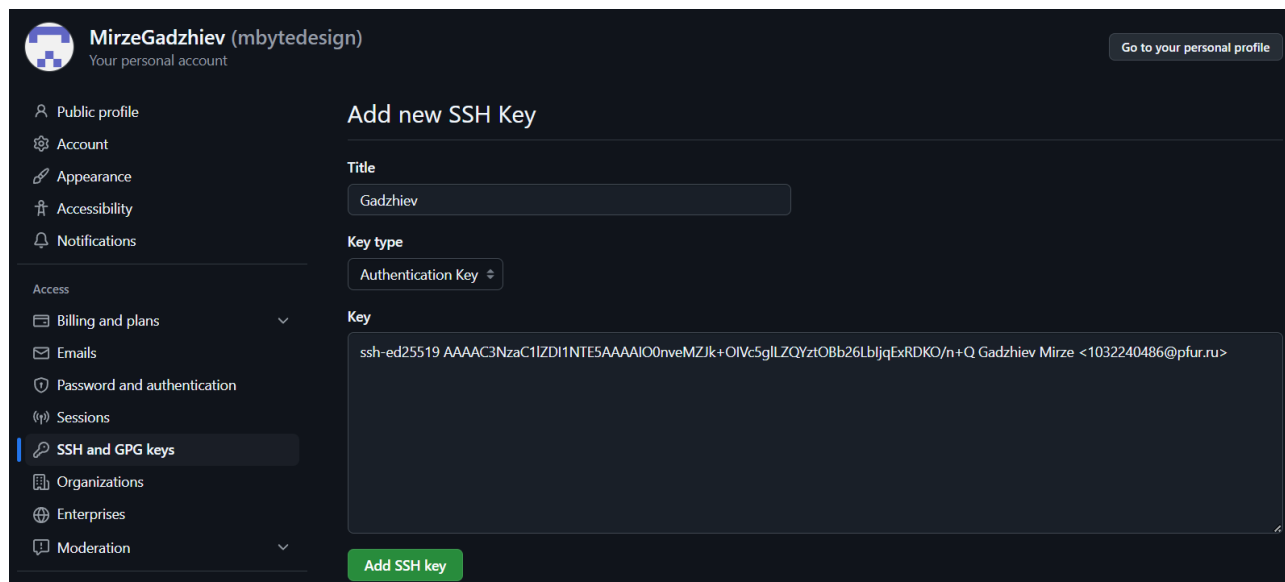


Рис. 7. Добавление ключа

4.4 Создание рабочего пространства и репозитория курса на основе шаблона

Создаю директорию, рабочее пространство, с помощью утилиты mkdir, благодаря ключу -p создаю все директории после домашней ~/work/study/2024-2025/«Архитектура компьютера»

```
liveuser@localhost-live:~$ mkdir -p ~/work/study/2024-2025/"Архитектура компьютера"
```

Рис. 8. Создание директории


В браузере перехожу на страницу репозитория с шаблоном курса по адресу <https://github.com/yamadharma/course-directory-student-template>. Далее выбираю «Use this template», чтобы использовать этот шаблон для своего репозитория. В открывшемся окне задаю имя репозитория (Repository name): study_2024–2025_arhpc и создаю репозиторий, нажимаю на кнопку «Create repository from template».

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


 **yamadharm/course-directory-student-template** ▾

Start your repository with a template repository's contents.

☐ **Include all branches**

Copy all branches from yamadharm/course-directory-student-template and not just the default branch.

Owner *

 **mbytedesign** ▾

Repository name *

/ **study_2024-2025_arhpc**

✓ Your new repository will be created as **study_2024-2025_arhpc**.

The repository name can only contain ASCII letters, digits, and the characters `.`, `-`, and `_`.

Great repository names are short and memorable. Need inspiration? How about **musical-octo-spork** ?

Description (optional)



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.

Рис. 9. Создание репозитория

Клонирую созданный репозиторий с помощью команды `git clone --recursive git@github.com:<user_name>/study_2024-2025_arhpc.git arch-pc`.

```
liveuser@localhost-live:~/work/study/2024-2025/Архитектура компьютера$ git clone --recursive git@github.com:mbytedesign/study_2024-2025_arhpc arch-pc
Cloning into 'arch-pc'...
remote: Enumerating objects: 33, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (32/32), done.
remote: Total 33 (delta 1), reused 18 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (33/33), 18.81 KiB | 344.00 KiB/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/liveuser/work/study/2024-2025/Архитектура компьютера/arch-pc/template/presentation'...
remote: Enumerating objects: 111, done.
remote: Counting objects: 100% (111/111), done.
remote: Compressing objects: 100% (77/77), done.
remote: Total 111 (delta 42), reused 100 (delta 31), pack-reused 0 (from 0)
Receiving objects: 100% (111/111), 102.17 KiB | 604.00 KiB/s, done.
Resolving deltas: 100% (42/42), done.
Cloning into '/home/liveuser/work/study/2024-2025/Архитектура компьютера/arch-pc/template/report'...
remote: Enumerating objects: 142, done.
remote: Counting objects: 100% (142/142), done.
remote: Compressing objects: 100% (97/97), done.
remote: Total 142 (delta 60), reused 121 (delta 39), pack-reused 0 (from 0)
Receiving objects: 100% (142/142), 341.09 KiB | 752.00 KiB/s, done.
Resolving deltas: 100% (60/60), done.
Submodule path 'template/presentation': checked out 'c9b2712b4b2d431ad5086c9c72a02bd2fca1d4a6'
Submodule path 'template/report': checked out 'c26e22effe7b3e0495707d82ef561ab185f5c748'
```

Рис. 10. Копирование репозитория

4.5 Настройка каталога курса

Перехожу в каталог `arch-pc` с помощью утилиты `cd`, удаляю лишние файлы с помощью утилиты `rm`.


```
liveuser@localhost-live:~/work/study/2024-2025/Архитектура компьютера$ cd ~/work/study/2024-2025/"Архитектура компьютера"/arch-pc
liveuser@localhost-live:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ rm package.json
```

Рис. 11. Удаление лишних файлов

Создаю каталог, отправляю каталоги с локального репозитория

```
liveuser@localhost-live:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ echo arch-pc > COURSE
liveuser@localhost-live:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git add .
liveuser@localhost-live:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git commit -am 'feat(main): make course structure'
[master e153730] feat(main): make course structure
 2 files changed, 1 insertion(+), 14 deletions(-)
 delete mode 100644 package.json
liveuser@localhost-live:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 289 bytes | 289.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:mbytedesign/study_2024-2025_arhpc
 5235492..e153730 master -> master
```

Рис. 12. Отправление изменений на сервер

Проверка изменений на странице репозитория

study_2024-2025_arhpc / labs /

Gadzhiev Mirze feat(main): make course structure

Name	Last commit message
..	
lab01	feat(main): make course structure
lab02	feat(main): make course structure

Рис. 13. Проверка изменений на странице репозитория

5.Выполнение заданий для самостоятельной работы

Перехожу в директорию labs/lab02/report с помощью утилиты cd. Создаю в каталоге файл для отчета по второй лабораторной работе с помощью утилиты touch. Я добавил отчет по предыдущей лабораторной работе в соответствующий каталог

study_2024-2025_arhpc / labs / lab01 / report /

Gadzhiev Mirze feat(main): make course structure

Name
..
Л01_Гаджиев_отчет.pdf

6 Выводы

При выполнении данной лабораторной работы я изучила идеологию и применение средств контроля версий, а также приобрела практические навыки по работе с системой git.

7 Вопросы для самопроверки

1 Системы контроля версий (VCS) — это программное обеспечение, которое помогает отслеживать изменения в файловой системе и эффективно управлять версиями файлов и кода в проекте. Они предназначены для решения следующих задач: Отслеживание изменений: позволяет отслеживать все изменения, сделанные в файлах проекта, включая добавление, удаление, изменение строк кода и текстовых данных. Версионирование: сохраняет изменения в виде версий, что позволяет восстанавливать предыдущие состояния проекта, откатывать изменения и переходить к определённым версиям для просмотра или восстановления кода. Ветвление и слияние (Branching and Merging): позволяет создавать отдельные ветки проекта, где разработчики могут работать независимо, а затем объединять свои изменения в основную ветку. Работа в команде: позволяет нескольким разработчикам работать над одним проектом, автоматически обнаруживая и решая конфликты при слиянии изменений. История изменений: подробно записывает все изменения, включая информацию о коммитах, авторах и времени внесения изменений.

2 Хранилище (репозиторий) — это специальное место, где хранятся файлы и папки проекта. Изменения в этих файлах отслеживаются системой контроля версий (VCS). Рабочая копия — это копия проекта, с которой разработчик работает напрямую. Он вносит изменения в рабочую копию, а затем периодически синхронизирует её с хранилищем. Синхронизация включает отправку изменений, сделанных разработчиком, в хранилище (commit) и актуализацию рабочей копии с последней версией из репозитория (update). История — это последовательность всех изменений, которые были внесены в проект с момента его создания. Она содержит информацию о том, кто, когда и какие изменения внёс. Таким образом, хранилище служит местом хранения проекта, рабочая копия — инструментом для работы разработчика, а commit и update обеспечивают связь между ними и сохранение истории изменений.

3 Централизованные VCS — это системы контроля версий, в которых репозиторий проекта находится на сервере, доступ к которому осуществляется через клиентское приложение. Примеры централизованных VCS: CVS (Concurrent Versions System) и Subversion (SVN). Децентрализованные VCS (также называемые распределёнными системами контроля версий, DVCS) хранят копию репозитория у каждого разработчика, работающего с системой. Локальные репозитории периодически синхронизируются с центральным репозиторием. Примеры децентрализованных VCS: Git и Mercurial.

4 При единоличной работе с хранилищем в системе контроля версий (VCS) выполняются следующие действия: Создание новой папки с датой или пометкой для рабочей версии проекта. Копирование рабочей версии проекта в новую папку. Непосредственная работа с рабочей копией проекта. Периодическая синхронизация рабочей копии с репозиторием путём отправки изменений (commit) и актуализации рабочей копии с последней версией из репозитория (update).

5 Порядок работы с общим хранилищем VCS включает следующие этапы: Создание репозитория: разработчик создаёт новый репозиторий, используя команды git init, hg init или аналогичные в зависимости от используемой системы контроля версий (например, Git, Mercurial). Внесение изменений в файлы: разработчик вносит изменения в файлы проекта, например, новые функции, исправления ошибок или другие изменения. Добавление файлов: разработчик добавляет файлы проекта в список подготовленных для сохранения в

репозитории с помощью команды `git add`, `hg add` или аналогичной. Коммит изменений: разработчик фиксирует изменения в файлах, создавая коммит с помощью команды `git commit`, `hg commit` или аналогичной. В коммите указывается, какие файлы были изменены, и краткое описание изменений. Отправка изменений на сервер: разработчик отправляет изменения на сервер с помощью команд `git push`, `hg push` или аналогичных.

6 Основные задачи, решаемые инструментальным средством Git: Возврат к любой предыдущей версии кода. Просмотр истории изменений. Параллельная работа над проектом. Backup кода.

7 `git clone` — клонирование репозитория в новую директорию. `git add` — перенос новых и изменённых файлов в проиндексированные. `git push` — отправка закоммиченных файлов в удалённый репозиторий. `git pull` — извлечение и загрузка последней информации в локальный репозиторий. `git rm` — удаление файла из удалённого репозитория.

8 Вот несколько примеров использования локального репозитория: Создание почтового аккаунта с использованием локального репозитория для хранения данных. Загрузка содержимого сайта с использованием локального репозитория и менеджера файлов. Настройка доступа к виртуальным папкам с помощью локального репозитория. Управление базами данных приложений с использованием локального репозитория. Примеры использования удалённого репозитория: Разработка программного обеспечения в команде: разработчики могут совместно работать над одним проектом, синхронизируя свои изменения с общим удалённым репозиторием. Хранение и обмен кодом: разработчики могут делиться своим кодом с другими участниками команды или сообществом, загружая его в удалённый репозиторий.