

# 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: Платонов Иван Георгиевич

Группа: НММбд-03-24

МОСКВА

2024 г.

## Оглавление

Список иллюстраций.....	3
Список таблиц.....	4
1. Цели работы .....	5
2. Задание.....	5
3. Теоретическое введение .....	5
3.1. Системы контроля версий. Общие понятия .....	5
3.2. Система контроля версий git .....	6
3.3. Основные команды git.....	6
3.4. Стандартные процедуры работы при наличии центрального репозитория.....	6
4. Выполнение лабораторной работы .....	7
4.1. Техническое обеспечение .....	7
4.2. Регистрация на GitHub .....	8
4.3. Локальная настройка git.....	9
4.4. Создание SSH-ключа для авторизации на GitHub .....	10
4.5. Создание копии репозитория (форка – fork) на основе уже существующего репозитория.....	11
4.6. Настройка каталога курса .....	12
5. Задание для самостоятельной работы .....	13
6. Выводы .....	13
8. Список литературы .....	14

## Список иллюстраций

Рисунок 1. Установка git на Linux Ubuntu .....	8
Рисунок 2. Окно регистрации на GitHub .....	8
Рисунок 3. Окно после успешной регистрации на GitHub .....	9
Рисунок 4. Локальная настройка git .....	10
Рисунок 5. Установка xclip .....	10
Рисунок 6. Создание криптографической пары ключей .....	10
Рисунок 7. Копирование в буфер обмена созданного публичного ключа .....	11
Рисунок 8. Добавление ключа аутентификации на GitHub .....	11
Рисунок 9. Клонирование репозитория .....	12
Рисунок 10. Настройка каталога курса .....	12
Рисунок 11. Решение задания для самостоятельной работы .....	13

## Список таблиц

Таблица 1. Самые частые команды git.....	6
Таблица 2. Описание параметров git config .....	9

## 1. Цели работы

Целями данной работы являются теоретическое и практическое ознакомление с системой контроля версиями (VSC – Version control system), а также наработка навыков в работе VSC git в связке с git сервер провайдером GitHub.

## 2. Задание

1. [Техническое обеспечение;](#)
2. [Регистрация на GitHub;](#)
3. [Локальная настройка git;](#)
4. [Создание SSH-ключа для авторизации на GitHub;](#)
5. [Создание копии репозитория \(форка – fork\) на основе уже существующего репозитория;](#)
6. [Настройка каталога курса;](#)
7. [Задание для самостоятельной работы.](#)

## 3. Теоретическое введение

### 3.1. Системы контроля версий. Общие понятия

**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.

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.

### 3.2. Система контроля версий git

Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды git с различными опциями.

Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией.

### 3.3. Основные команды git

Наиболее часто используемые команды git представлены в таблице 1.

Таблица 1. Самые частые команды git

Команда	Описание
<i>git init</i>	создание основного дерева репозитория
<i>git pull</i>	получение обновлений (изменений) текущего дерева из центрального репозитория
<i>git push</i>	отправка всех произведённых изменений локального дерева в центральный репозиторий
<i>git status</i>	просмотр списка изменённых файлов в текущей директории
<i>git diff</i>	просмотр текущих изменений
<i>git add</i>	добавить все изменённые и/или созданные файлы и/или каталоги
<i>git add</i> <i>имена_файлов</i>	добавить конкретные изменённые и/или созданные файлы и/или каталоги
<i>git rm</i> <i>имена_файлов</i>	удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории)
<i>git commit -am</i> <i>'Описание коммита'</i>	сохранить все добавленные изменения и все изменённые файлы
<i>git checkout -b</i> <i>имя_ветки</i>	создание новой ветки, базирующейся на текущей
<i>git checkout</i> <i>имя_ветки</i>	переключение на некоторую ветку (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой)
<i>git push origin</i> <i>имя_ветки</i>	отправка изменений конкретной ветки в центральный репозиторий
<i>git merge --no-ff</i> <i>имя_ветки</i>	слияние ветки с текущим деревом
<i>git branch -d</i> <i>Команда</i>	удаление локальной уже слитой с основным деревом ветки
<i>git branch -D</i> <i>имя_ветки</i>	Описание принудительное удаление локальной ветки
<i>git push</i> <i>origin :имя_ветки</i>	удаление ветки с центрального репозитория

### 3.4. Стандартные процедуры работы при наличии центрального репозитория

Работа пользователя со своей веткой начинается с проверки и получения изменений

из центрального репозитория (при этом в локальное дерево до начала этой процедуры не должно было вноситься изменений):

***git checkout master***

***git pull***

***git checkout -b имя\_ветки***

Затем можно вносить изменения в локальном дереве и/или ветке. После завершения внесения какого-то изменения в файлы и/или каталоги проекта необходимо разместить их в центральном репозитории. Для этого необходимо проверить, какие файлы изменились к текущему моменту:

***git status***

и при необходимости удаляем лишние файлы, которые не хотим отправлять в центральный репозиторий.

Затем полезно просмотреть текст изменений на предмет соответствия правилам ведения чистых коммитов:

***git diff***

Если какие-либо файлы не должны попасть в коммит, то помечаем только те файлы, изменения которых нужно сохранить. Для этого используем команды добавления и/или удаления с нужными опциями:

***git add имена\_файлов***

***git rm имена\_файлов***

Если нужно сохранить все изменения в текущем каталоге, то используем:

***git add .***

Затем сохраняем изменения, поясняя, что было сделано:

***git commit -am "Some commit message"***

и отправляем в центральный репозиторий:

***git push origin имя\_ветки***

или

***git push***

## **4. Выполнение лабораторной работы**

### **4.1. Техническое обеспечение**

Так как работа производится на домашнем ПК, то необходимо установить git командой ***sudo apt install git*** (Linux Ubuntu).

```
eventgraph@igplatonov:~$ sudo apt install git
[sudo] пароль для eventgraph:
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Будут установлены следующие дополнительные пакеты:
  git-man liberror-perl
Предлагаемые пакеты:
  git-daemon-run | git-daemon-sysvinit git-doc git-email
  git-cvs git-mediawiki git-svn
Следующие HOBBIE пакеты будут установлены:
  git git-man liberror-perl
Обновлено 0 пакетов, установлено 3 новых пакетов, для у
Необходимо скачать 4 804 кВ архивов.
После данной операции объём занятого дискового простран
Хотите продолжить? [Д/н] у
Пол:1 http://ru.archive.ubuntu.com/ubuntu noble/main ar
Пол:2 http://ru.archive.ubuntu.com/ubuntu noble-updates
Пол:3 http://ru.archive.ubuntu.com/ubuntu noble-updates
Получено 4 804 кВ за 0с (28,7 MB/s)
Выбор ранее не выбранного пакета liberror-perl.
```

Рисунок 1. Установка git на Linux Ubuntu

## 4.2. Регистрация на GitHub

Рисунок 2. Окно регистрации на GitHub

Для того, чтобы зарегистрировать новый аккаунт, переходим на официальный веб-сайт GitHub: <https://github.com>, и нажимаем кнопку sign up. Далее заполняем форму, проходим опрос и получаем свой аккаунт. Однако стоит заметить, что, так как GitHub принадлежит Microsoft и Microsoft запретила регистрацию на почтовые аккаунты доменных зон России (\*.ru, \*.bk, \*.su и т.д.), то приходится регистрироваться на личную почту – на корпоративную почту РУДН в данный момент не получится.



После успешной регистрации появится окно нового аккаунта.

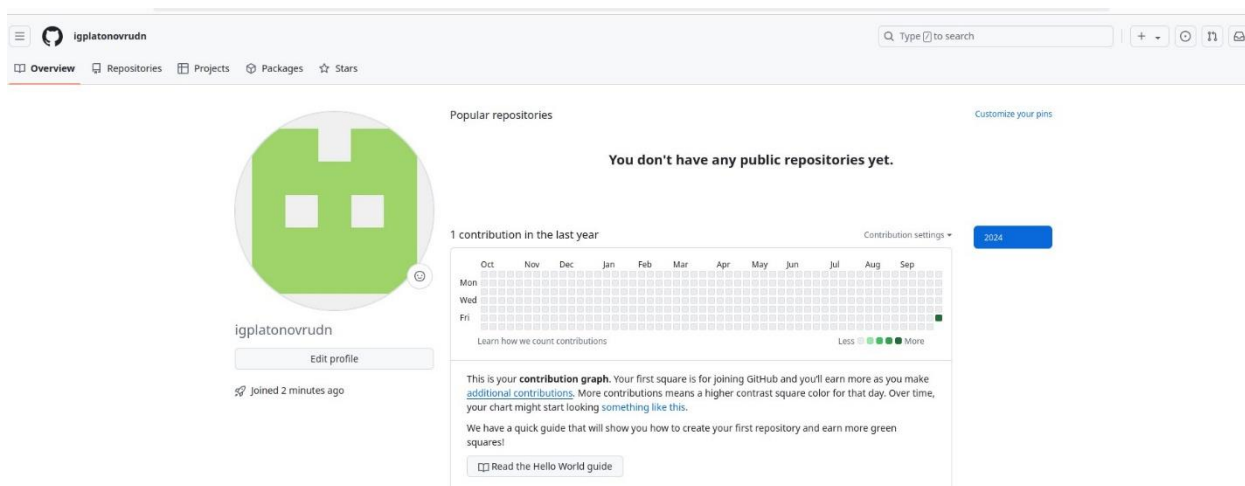


Рисунок 3. Окно после успешной регистрации на GitHub

### 4.3. Локальная настройка git

Настройка осуществляется при помощи редактирования файла конфигурации git путем ввода команды **git config**. Ключ **--global** означает редактирование общего конфига; если необходимо редактировать только конфиг отдельного репозитория, то стоит использовать ключ **--local**. Далее стоит рассмотреть изменяемые параметры

Таблица 2. Описание параметров git config

Параметр	Описание
<code>user.name</code>	имя пользователя, от кого будут публиковаться «коммиты» (commits)
<code>user.email</code>	электронная почта пользователя, от кого будут публиковаться «коммиты» (commits)
<code>init.defaultBranch</code>	название ветки по умолчанию
<code>core.quotePath</code>	необходимо ли обрамлять «необычные» символы в кавычки (такие как <code>\t</code> , <code>\n</code> и т.д.)
<code>core.autocrlf</code>	указывает на то, должны ли пути в директории оканчиваться на CRLF (символ каретки и символ перевода на новую строку)
<code>core.safecrlf</code>	проверяет значение <code>core.autocrlf</code> и может давать пользователю предупреждения об ошибках в случаях, когда <code>core.autocrlf</code> установлен на true

Для того, чтобы проверить установленные значения, можно воспользоваться командой **git config --list**.

```
eventgraph@igplatonov: ~  
eventgraph@igplatonov:~$ git config --global user.name "Platonov Ivan"  
eventgraph@igplatonov:~$ git config --global user.email "ivanplateam@gmail.com"  
eventgraph@igplatonov:~$ git config --global core.quotePath false  
eventgraph@igplatonov:~$ git config --global init.defaultBranch master  
eventgraph@igplatonov:~$ git config --global core.autocrlf input  
eventgraph@igplatonov:~$ git config --global core.safecrlf warn  
eventgraph@igplatonov:~$ git config --list  
user.name=Platonov Ivan  
user.email=ivanplateam@gmail.com  
core.quotePath=false  
core.autocrlf=input  
core.safecrlf=warn  
init.defaultBranch=master  
eventgraph@igplatonov:~$
```

Рисунок 4. Локальная настройка git

#### 4.4. Создание SSH-ключа для авторизации на GitHub

Для начала желательно установить *xclip*, чтобы копировать из файлов, не открывая их. Вводим команду *sudo apt install xclip* и получаем данный пакет себе на компьютер.

```
eventgraph@igplatonov:~/.ssh$ sudo apt install xclip  
[sudo] пароль для eventgraph:  
Чтение списков пакетов... Готово  
Построение дерева зависимостей... Готово  
Чтение информации о состоянии... Готово  
Следующие НОВЫЕ пакеты будут установлены:  
  xclip  
Обновлено 0 пакетов, установлено 1 новых пакетов, для удаления отмечено 0 пакетов, и 25 пак  
Необходимо скачать 17,6 kB архивов.  
После данной операции объем занятого дискового пространства возрастет на 54,3 kB.
```

Рисунок 5. Установка xclip

Далее при помощи команды *ssh-keygen* с указанием имени и почты пользователя создаем публичный и приватный ключи шифрования. Публичный ключ будет иметь расширение \*.pub. Ключи создаются по умолчанию при помощи алгоритма шифрования EdDSA (Ed25519). Публичный ключ имеет длину 256 бит, а приватный – 512 бит.

```
eventgraph@igplatonov:~$ ssh-keygen -C "Platonov Ivan ivanplateam@gmail.com"  
Generating public/private ed25519 key pair.  
Enter file in which to save the key (/home/eventgraph/.ssh/id_ed25519):  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/eventgraph/.ssh/id_ed25519  
Your public key has been saved in /home/eventgraph/.ssh/id_ed25519.pub  
The key fingerprint is:
```

Рисунок 6. Создание криптографической пары ключей

После чего при помощи команд *cat* и *xclip* копируем содержимое публичного ключа в буфер обмена.

```
eventgraph@igplatonov:~/.ssh$ cat ~/.ssh/id_ed25519.pub | xclip -sel clip
eventgraph@igplatonov:~/.ssh$
```

Рисунок 7. Копирование в буфер обмена созданного публичного ключа

После копирования переходим на страницу добавления нового SSH ключа в GitHub. Даем ему какое-то имя, тип выставляем на ключ аутентификации и вставляем его из буфера обмена в соответствующее поле.

Рисунок 8. Добавление ключа аутентификации на GitHub

#### 4.5. Создание копии репозитория (форка – fork) на основе уже существующего репозитория

Для того, чтобы создать форк репозитории, переходим по URL <https://github.com/yamadharm/course-directory-student-template> и нажимаем кнопку “Use this template -> Create a new repository”. После чего репозитория появится у нас на аккаунте. Далее командой **cd** переходим по рабочей директории

```
cd ~/work/study/2024–2025/"Архитектура компьютера"
```

и клонируем нашу репозиторию

```
git clone --recursive git@github.com:/study_2024–2025_arh-pc.git arch-pc
```

```

eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера$ git clone --recursive git@github.com:igplatonovrudn/study_2024-2025_arh-pc.git
Клонирование в «study_2024-2025_arh-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)
Получение объектов: 100% (33/33), 18.81 КиБ | 4.70 МиБ/с, готово.
Определение изменений: 100% (1/1), готово.
Подмодуль «template/presentation» (https://github.com/yamadharma/academic-presentation-markdown-template.git) зарегистрирован по пути «template/presentation»
Подмодуль «template/report» (https://github.com/yamadharma/academic-laboratory-report-template.git) зарегистрирован по пути «template/report»
Клонирование в «/home/eventgraph/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-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)
Получение объектов: 100% (111/111), 102.17 КиБ | 1.05 МиБ/с, готово.
Определение изменений: 100% (42/42), готово.
Клонирование в «/home/eventgraph/work/study/2024-2025/Архитектура компьютера/study_2024-2025_arh-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)
Получение объектов: 100% (142/142), 341.09 КиБ | 2.38 МиБ/с, готово.
Определение изменений: 100% (60/60), готово.
Submodule path 'template/presentation': checked out 'c9b2712b4b2d431ad5086c9c72a02bd2fca1d4a6'
Submodule path 'template/report': checked out 'c26e22effe7b3e0495707d82ef561ab185f5c748'
eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера$

```

Рисунок 9. Клонирование репозитория

## 4.6. Настройка каталога курса

Переходим в каталог курса, далее удаляем файл `package.json` и создаем файл `arch-pc` с содержанием «COURSE», после чего добавляем все изменения в локальный коммит и делаем пуш на сервер.

```

cd ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc
rm package.json
echo arch-pc > COURSE
git add .
git commit -am 'feat(main): make course structure'
git push

```

```

eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git add .
eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git checkout
M
  COURSE
D
  package.json
Эта ветка соответствует «origin/master».
eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git commit -am feat(main): make course structure
bash: синтаксическая ошибка рядом с неожиданным маркером «(»
eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git commit -am'feat(main): make course structure'
[master 768c1a9] feat(main): make course structure
 2 files changed, 1 insertion(+), 14 deletions(-)
 delete mode 100644 package.json
eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git push
Перечисление объектов: 5, готово.
Подсчет объектов: 100% (5/5), готово.
При сжатии изменений используется до 20 потоков
Сжатие объектов: 100% (2/2), готово.
Запись объектов: 100% (3/3), 291 байт | 291.00 КиБ/с, готово.
Всего 3 (изменений 1), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:igplatonovrudn/study_2024-2025_arh-pc-.git
 4652a89..768c1a9  master -> master
eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера/arch-pc$

```

Рисунок 10. Настройка каталога курса



## 5. Задание для самостоятельной работы

### Формулировка задания:

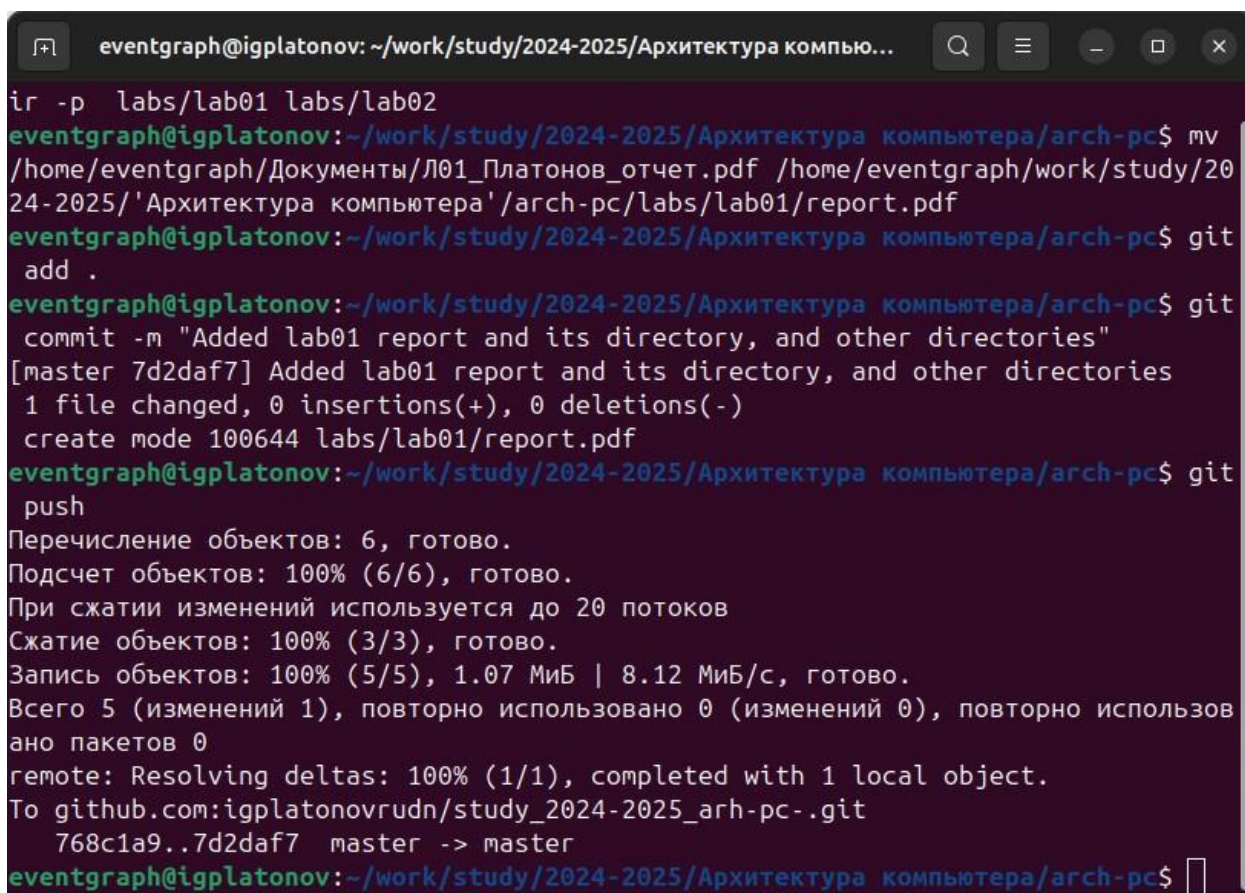
«

1. Создайте отчет по выполнению лабораторной работы в соответствующем каталоге рабочего пространства (**labs>lab02>report**).
2. Скопируйте отчеты по выполнению предыдущих лабораторных работ в соответствующие каталоги созданного рабочего пространства.
3. Загрузите файлы на github.

»

### Решение:

Тут все достаточно просто: создаем нужные директории командой **mkdir**, переносим файл с отчетом за первую лабораторную работу командой **mv**, при этом изменив ему имя, добавляем все в локальный коммит и делаем пуш.



```
eventgraph@igplatonov: ~/work/study/2024-2025/Архитектура компью...
ir -p labs/lab01 labs/lab02
eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ mv
/home/eventgraph/Документы/Л01_Платонов_отчет.pdf /home/eventgraph/work/study/20
24-2025/'Архитектура компьютера'/arch-pc/labs/lab01/report.pdf
eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git
add .
eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git
commit -m "Added lab01 report and its directory, and other directories"
[master 7d2daf7] Added lab01 report and its directory, and other directories
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 labs/lab01/report.pdf
eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ git
push
Перечисление объектов: 6, готово.
Подсчет объектов: 100% (6/6), готово.
При сжатии изменений используется до 20 потоков
Сжатие объектов: 100% (3/3), готово.
Запись объектов: 100% (5/5), 1.07 МиБ | 8.12 МиБ/с, готово.
Всего 5 (изменений 1), повторно использовано 0 (изменений 0), повторно использов
ано пакетов 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:igplatonovrudn/study_2024-2025_arh-pc-.git
768c1a9..7d2daf7 master -> master
eventgraph@igplatonov:~/work/study/2024-2025/Архитектура компьютера/arch-pc$
```

Рисунок 11. Решение задания для самостоятельной работы

## 6. Выводы

VCS git – одна из самых необходимых вещей в разработке. Именно с ее помощью большие команды разработчиков могут вместе работать над одним проектом. Стоит также заметить, что с помощью git и GitHub в связке можно не только работать одновременно над одним проектом, но и создавать сложные Developers Operations (DevOps), настраивать CI/CD операции и pipeline рабочего проекта, однако это уже не тема данной лабораторной работы. В данной лабораторной работе были изучены и проделаны базовые операции работы с git и GitHub, что, естественно, очень полезно для студента, не имевшего раньше опыт работы с данной VCS.

## 8. Список литературы

- Edwards-Curve Digital Signature Algorithm (EdDSA)*, Internet Research Task Force (IRTF) . (1 2017 г.). Получено из datatracker: <https://datatracker.ietf.org/doc/html/rfc8032>
- Software Freedom Conservancy Inc. (2024). *git documentation*. Получено из git-scm: <https://git-scm.com/docs/git-config>
- Демидова А. В. (б.д.). Лабораторная работа №1. Основы интерфейса командной строки ОС GNU Linux. В Д. А.В., *Архитектура ЭВМ* (стр. 1-13). Москва.
- Колисниченко Д.Н. (2023). *Командная строка Linux*. Санкт-Петербург: ВХВ-Петербург.
- Левицкий , Д. Н., & Завьялов, А. В. (2023). *Сервер на Windows и Linux. Администрирование и виртуализация*. Санкт-Петербург: Издательство Наука и Техника.
- Фишерман, Л. В. (2022). *Git. Практическое руководство. Управление и контроль версий в разработке программного обеспечения*. Санкт-Петербург: Издательство "Наука и Техника".
- Чакон, С., & Штрауб, Б. (2022). *Git для профессионального разработчика*. Санкт-Петербург: Издание "Питер".