

**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: Земцов Роман

Группа: НКАбд-02-25

**МОСКВА**

2025 г.

## Содержание

Цель работы

2.2. Теоретическое введение

2.2.1. Системы контроля версий. Общие понятия

2.2.2. Система контроля версий Git

2.2.3. Основные команды git

Порядок выполнения лабораторной работы

2.4.1. Настройка github

2.4.2. Базовая настройка git

2.4.3. Создание SSH-ключа

2.4.4. Создание рабочего пространства и репозитория курса на основе шаблона

2.4.5. Создание репозитория курса на основе шаблона

2.4.6. Настройка каталога курса

Выполнение самостоятельной работы

Выводы

## **Цель работы**

Целью работы является изучение идеологии и применения средств контроля версий, приобретение практических навыков по работе с системой контроля версий git

## 2.2 Теоретическое введение

### 2.2.1. Системы контроля версий. Общие понятия.

**Система управления версиями** (также используется определение «система контроля версий<sup>[1]</sup>», от [англ.](#) *version control system*, *VCS* или *revision control system*) — [программное обеспечение](#) для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при [разработке программного обеспечения](#) для хранения [исходных кодов](#) разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в [САПР](#), обычно в составе систем управления данными об изделии ([PDM](#)). Управление версиями используется в инструментах [конфигурационного управления](#) (*Software Configuration Management Tools*).

Программное обеспечение [Википедии](#) ведёт историю изменений для всех её статей, используя методы, аналогичные тем, которые применяются в системах управления версиями.

### 2.2.2. Система контроля версий Git.

**Git** (произносится «гит»<sup>[9]</sup>) — распределённая [система управления версиями](#). Проект был создан [Линусом Торвальдсом](#) для управления разработкой [ядра Linux](#), первая версия выпущена 7 апреля 2005 года; координатор — [Джон Хамано](#).

Среди проектов, использующих Git, — [ядро Linux](#), [Swift](#), [Android](#), [Drupal](#), [Cairo](#), [GNU Core Utilities](#), [Mesa](#), [Wine](#), [Chromium](#), [Compiz](#), [Fusion](#), [FlightGear](#), [jQuery](#), [PHP](#), [NASM](#), [MediaWiki](#), [DokuWiki](#), [Qt](#), ряд дистрибутивов [Linux](#).

Программа является свободной и выпущена под лицензией [GNU GPL](#) версии 2. По умолчанию используется [TCP-порт](#) 9418.

Система спроектирована как набор программ, специально разработанных с учётом их использования в [сценариях](#). Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы.

Например, [Cogito](#) является именно таким примером оболочки к репозиториям Git, а [StGit](#) использует Git для управления коллекцией исправлений ([патчей](#)).

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и [Darcs](#), [BitKeeper](#), [Mercurial](#), [Bazaar](#) и [Monotone](#)<sup>[англ.]</sup>, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удалённый доступ к репозиториям Git обеспечивается git-[демоном](#), [SSH](#)- или [HTTP](#)-сервером. TCP-сервис git-daemon входит в дистрибутив Git и, наряду с SSH, является наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

### 2.2.3. Основные команды Git.

1. Клонирование репозитория — `git clone https://github.com/<Ваш аккаунт>/<Ваш репозиторий>.git` .
2. Список имён удалённых репозиторияев с URL-ом — `git remote -v`
3. Добавление файлов в индекс — `git add` .
4. Удаление файлов из индекса — `git reset`
5. Создание коммита — `git commit -m "Сообщение коммита"`
6. Добавить забытые файлы в предыдущий коммит без изменения сообщения:

```
git add .  
git commit --amend --no-edit
```

- 7.Отправить изменения в удаленный репозиторий — `git push`
- 8.Создание ветки и переключение на нее — `git switch -c <новая_ветка>`
- 9.Просмотр локальных и удалённых веток — `git branch -a`
- 10.Принудительное удаление ветки — `git branch -D <ветка>`
- 11.Переименовать ветку — `git branch --move (-m) <старая_ветка> <новая_ветка>`
- 12.Состояние рабочего каталога и индекса — `git status`
- 13.Просмотр истории коммитов в удобном виде — `git log --graph --oneline`
- 14.Восстановление всех файлов, не вошедших в индекс, до состояния последнего коммита — `git checkout -- .`
- 15.Убирает все файлы из индекса — `git restore --staged ..` - все файлы, можно заменить на название определенного файла.
- 16.Слияние изменений веток — `git merge --no-ff <ветка, которую хотим слить в текущую> -m "<Сообщение>"`
- 17.Извлечение и слияние изменений — `git pull`
- 18.Удаление файла из индекса (staging area) и из remote репозитория при следующем push, но сохранение в рабочей директории — `git rm --cached <имя-файла>`. -r флаг действует для папок.
- 19.Если необходимо откатить коммит и убрать его из истории, то просто делаем следующие шаги:

```
git log --oneline
3dd9bb9 (HEAD -> master, origin/master, origin/HEAD) Cherry-pick from develop
60c5923 Initial commit

git reset --hard 60c5923
HEAD is now at 60c5923 Initial commit

git push -f
```

## 2.2.4. Стандартные процедуры работы при наличии центрального репозитория

**Репозиторий программного обеспечения**, или сокращённо **репозиторий**, — это место хранения [программных пакетов](#). Часто вместе с метаданными хранится оглавление. Репозиторий программного обеспечения обычно управляется исходным кодом или [системой контроля версий](#), либо менеджерами репозитория. [Менеджеры пакетов](#) позволяют автоматически устанавливать и обновлять репозитории, которые иногда называют «пакетами».

Многие издатели программного обеспечения и другие организации предоставляют для этой цели серверы в [Интернете](#) бесплатно или за абонентскую плату. Репозитории могут быть предназначены только для конкретных программ, таких как [CPAN](#) для [Perl языка программирования](#), или для целой [операционной системы](#). Операторы таких репозитория обычно предоставляют [систему управления пакетами](#) — инструменты для поиска, установки и других манипуляций с пакетами программного обеспечения из репозитория. Например, во многих [дистрибутивах Linux](#) используется [Advanced Packaging Tool](#) (APT), который обычно встречается в дистрибутивах на основе [Debian](#), или Yellowdog Updater, Modified ([yum](#)), который встречается в дистрибутивах на основе [Red Hat](#). Существует также множество независимых систем управления пакетами, таких как [raspm](#), используемая в [Arch](#)

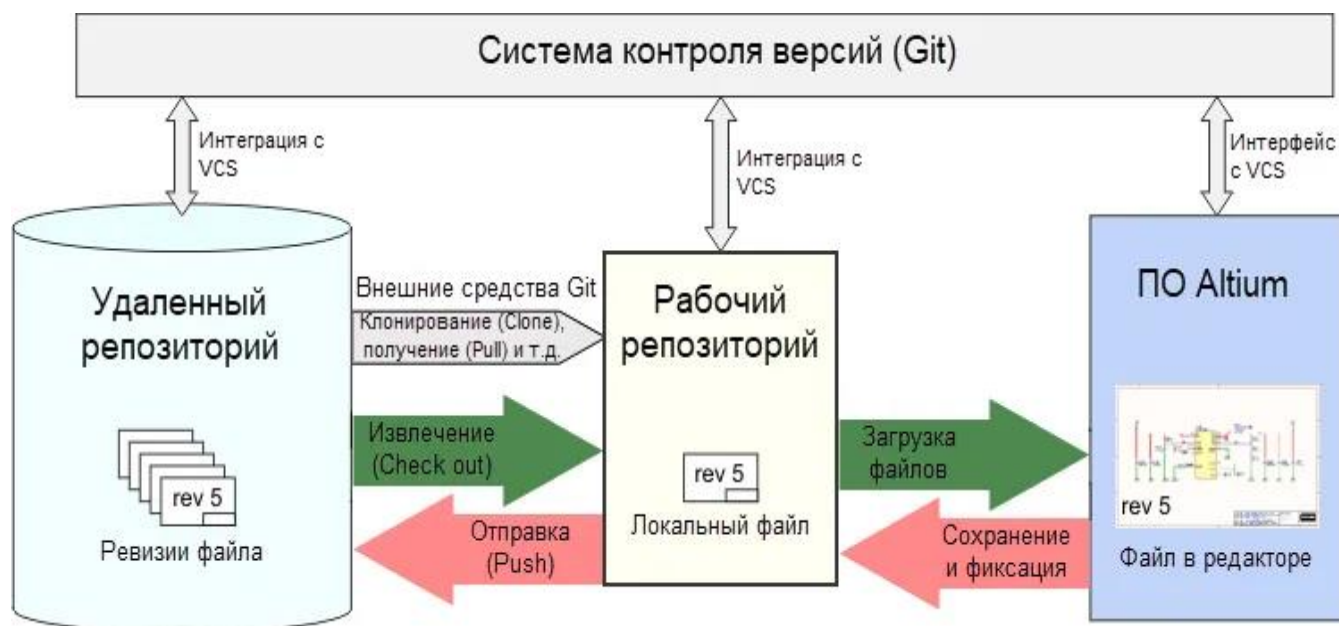
[Linux](#), и equo, встречающаяся в [Sabayon Linux](#).

Пример подписанного ключа репозитория (с [ZYpp](#) на [openSUSE](#))

Поскольку репозитории программного обеспечения предназначены для хранения полезных пакетов, основные репозитории защищены от [вредоносного ПО](#). Если компьютер настроен на использование [подписанного цифровой подписью](#) репозитория от надёжного поставщика и оснащён соответствующей [системой разрешений](#), это значительно снижает риск заражения таких систем вредоносным ПО. Кроме того, многие системы с такими возможностями не нуждаются в программах для защиты от вредоносного ПО, таких как [антивирусное ПО](#).<sup>[1]</sup>

Большинство основных [дистрибутивов Linux](#) имеют множество репозиториях по всему миру, которые являются зеркалами основного репозитория.

На стороне клиента менеджер пакетов помогает устанавливать и обновлять пакеты из репозиториях.



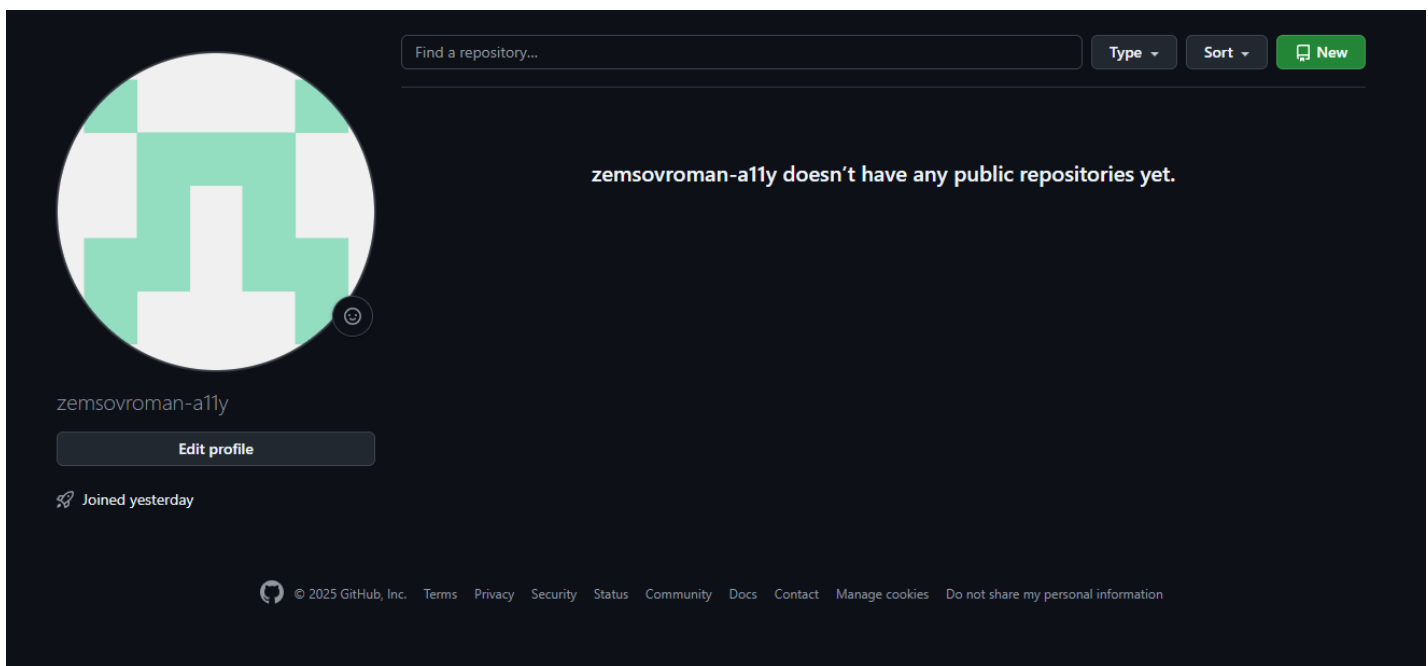
## Выполнение лабораторной работы

### Настройка github

Для начала нужно создать учётную запись на сайте GitHub, который будет использоваться как удаленный сервер для хранения репозиториев

### Инструкция по выполнению

1. Перейдите на сайт <https://github.com/>.
2. Нажмите на кнопку "Sign up".
3. Следуйте инструкциям на экране: введите вашу электронную почту, создайте пароль и выберите имя пользователя.
4. Подтвердите свою учетную запись через письмо, которое придет на указанную почту



Я успешно зарегистрировался на GitHub.

## 3.2 Базовая настройка git

После установки Git на локальной машине необходимо было выполнить базовую конфигурацию: указать имя пользователя и адрес электронной почты, которые будут отображаться в истории коммитов, а также настроить другие параметры для корректной работы.

```
(rzemcov@rzemcov)-[~]
$ git config --global user.name "Roman Zemcov"

(rzemcov@rzemcov)-[~]
$ git config --global user.email "1032254195@pfur.ru"

(rzemcov@rzemcov)-[~]
$ ^[[200~git config --global core.quotePath false
zsh: substitution failed

(rzemcov@rzemcov)-[~]
$ git config --global init.defaultBranch master

(rzemcov@rzemcov)-[~]
$ git config --global core.autocrlf input

(rzemcov@rzemcov)-[~]
$ git config --global core.quotePath false

(rzemcov@rzemcov)-[~]
$ git config --global core.safecrlf warn

(rzemcov@rzemcov)-[~]
$
```

Я выполнил базовую настройку Git. Эти команды нужны, чтобы каждый мой коммит (сохранение изменений) был подписан моим именем и почтой. Это очень важно для отслеживания истории изменений, особенно при работе в команде.

## 3.3 Создание SSH-ключа

Для безопасного подключения к GitHub без необходимости каждый раз вводить пароль, я сгенерировал пару SSH-ключей (приватный и публичный) и добавил публичный ключ в свой профиль на GitHub.

```

(rzemcov@rzemcov)-[~]
$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/rzemcov/.ssh/id_ed25519):
Created directory '/home/rzemcov/.ssh'.
Enter passphrase for "/home/rzemcov/.ssh/id_ed25519" (empty for no passphrase
):
Enter same passphrase again:
Your identification has been saved in /home/rzemcov/.ssh/id_ed25519
Your public key has been saved in /home/rzemcov/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:n2lWsDEHVDYPbJza9If1XFVi4JZmfdRB6zjFRuhIXow rzemcov@rzemcov
The key's randomart image is:
+--[ED25519 256]--+
|      .o+Oo=+O|
|      E*@oo=|
|      *=@.oO+|
|      .%..B.+|
|      S . .o o|
|      . + .|
|      *|
|      o|
+-----[SHA256]-----+

```

Мы создали ключ, теперь его можно скопировать данной командой:

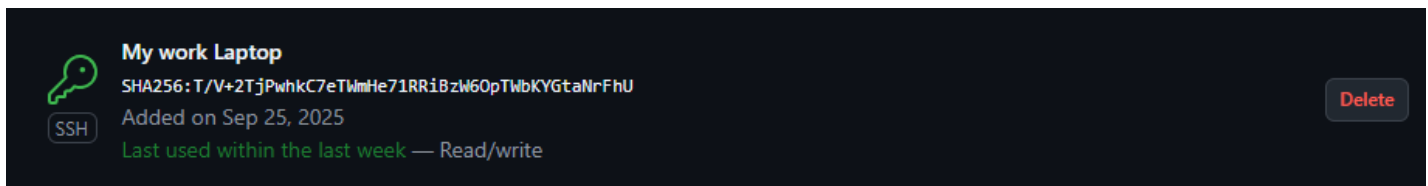
```

(rzemcov@rzemcov)-[~]
$ cat ~/.ssh/id_ed25519.pub

```

Теперь давайте добавим ключ на GitHub:

1. Зайдем в свой профиль на GitHub.
2. Перейдем в Settings -> SSH and GPG keys.
3. Нажмем New SSH key.
4. В поле Title введем название ключа (например, "My Work Laptop"), а в поле Key вставим скопированный ключ.
5. Нажмите Add SSH key.



Я сгенерировал SSH-ключ. Это позволяет мне безопасно подключаться к моему репозиторию на GitHub. Публичный ключ я добавил в настройки аккаунта, а приватный остался на моем компьютере, что обеспечивает безопасность соединения.

### 3.4 Создание рабочего пространства и репозитория курса

На этом этапе я создал репозиторий на GitHub на основе предоставленного шаблона, затем создал локальную структуру каталогов для учебных проектов и клонировал удаленный репозиторий на свой компьютер.

1. Я перешел на страницу репозитория с шаблоном курса:  
<https://github.com/yamadharm/course-directory-student-template>.
2. Нажал на кнопку **Use this template**.
3. В открывшемся окне задал имя репозитория (**Repository name**) **study\_2025–2026\_arch-pc** и создал репозиторий, нажав кнопку **Create repository from template**.

Теперь создадим каталог и перейдем в него:

```
(rzemcov@rzemcov)-[~]  
$ mkdir -p ~/work/study/2025-2026/"Архитектура компьютера"
```

```
(rzemcov@rzemcov)-[~]  
$ cd ~/work/study/2025-2026/"Архитектура компьютера"/
```

Клонируем созданный удаленный репозиторий на свой компьютер, скопировав SSH-ссылку со страницы репозитория.

```

(rzemcov@rzemcov)-[~/work/study/2025-2026/Архитектура компьютера]
$ git clone --recursive git@github.com:zemsovroman-ally/study_arch-pc.git
Cloning into 'study_arch-pc' ...
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 38 (delta 1), reused 26 (delta 1), pack-reused 0 (from 0)
Receiving objects: 100% (38/38), 23.45 KiB | 64.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.
Submodule 'template/presentation' (https://github.com/yamadharma/academic-presentation-markdown-template.git) registered for path 'template/presentation'
Submodule 'template/report' (https://github.com/yamadharma/academic-laboratory-report-template.git) registered for path 'template/report'
Cloning into '/home/rzemcov/work/study/2025-2026/Архитектура компьютера/study_arch-pc/template/presentation' ...
remote: Enumerating objects: 161, done.
remote: Counting objects: 100% (161/161), done.
remote: Compressing objects: 100% (111/111), done.
remote: Total 161 (delta 60), reused 142 (delta 41), pack-reused 0 (from 0)
Receiving objects: 100% (161/161), 2.65 MiB | 634.00 KiB/s, done.
Resolving deltas: 100% (60/60), done.
Cloning into '/home/rzemcov/work/study/2025-2026/Архитектура компьютера/study_arch-pc/template/report' ...
remote: Enumerating objects: 221, done.
remote: Counting objects: 100% (221/221), done.
remote: Compressing objects: 100% (152/152), done.
remote: Total 221 (delta 98), reused 180 (delta 57), pack-reused 0 (from 0)
Receiving objects: 100% (221/221), 765.46 KiB | 537.00 KiB/s, done.
Resolving deltas: 100% (98/98), done.
Submodule path 'template/presentation': checked out '6efd5c4ee78e4456caff3dc7062cfcad26058ca6'
Submodule path 'template/report': checked out '89a9622199b4df88227b9b3fa3d4714c85f68dd2'

```

Сначала я создал репозиторий на **GitHub**, используя веб-интерфейс и готовый шаблон. Затем я создал локальную структуру папок на своем компьютере и клонировал туда удаленный репозиторий. Теперь у меня есть локальная копия проекта, которая связана с сервером на **GitHub**

### 3.5 Настройка каталога курса

Завершающим этапом основной части работы была настройка структуры каталога курса с помощью **make** и отправка начальных файлов на сервер **GitHub**

```

(rzemcov@rzemcov)-[~/work/study/2025-2026/Архитектура компьютера]
$ cd ~/work/study/2025-2026/"Архитектура компьютера"/study_arch-pc/

(rzemcov@rzemcov)-[~/../study/2025-2026/Архитектура компьютера/study_arch-pc]
$ echo study_arch-pc > COURSE

(rzemcov@rzemcov)-[~/../study/2025-2026/Архитектура компьютера/study_arch-pc]
$ make prepare

```

Сейчас я перешел в каталог с проектом и выполнил подготовку структур

```
(rzemcov@rzemcov)-[~]
$ cd ~/work/study/2025-2026/"Архитектура компьютера"/study_arch-pc

(rzemcov@rzemcov)-[~/.../study/2025-2026/Архитектура компьютера/study_arch-pc]
$ echo arch-pc > COURSE

(rzemcov@rzemcov)-[~/.../study/2025-2026/Архитектура компьютера/study_arch-pc]
$ make prepare
```

```
$ cd /home/rzemcov/work/study/2025-2026/"Архитектура компьютера"/study_arch-pc

(rzemcov@rzemcov)-[~/.../study/2025-2026/Архитектура компьютера/study_arch-pc]
$ git add .

(rzemcov@rzemcov)-[~/.../study/2025-2026/Архитектура компьютера/study_arch-pc]
$ git commit -am 'feat(main): make course structure'
[master 096b5e0] feat(main): make course structure
212 files changed, 8074 insertions(+), 208 deletions(-)
```

```
(rzemcov@rzemcov)-[~/.../study/2025-2026/Архитектура компьютера/study_arch-pc]
$ git push
Enumerating objects: 68, done.
Counting objects: 100% (68/68), done.
Compressing objects: 100% (52/52), done.
Writing objects: 100% (65/65), 700.11 KiB | 1.31 MiB/s, done.
Total 65 (delta 23), reused 1 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (23/23), completed with 1 local object.
To github.com:zemsovroman-a1ly/study_arch-pc.git
 97ee7e3..096b5e0  master → master
```

Я завершил настройку структуры курса, а затем использовал основной рабочий цикл Git: добавил изменения в индекс (**git add**), зафиксировал их с осмысленным комментарием (**git commit**) и отправил на удаленный сервер (**git push**). Теперь все мои локальные изменения синхронизированы с **GitHub**.

### Выполнение самостоятельной работы

Задание №1. Создайте отчет по выполнению лабораторной работы в соответствующем каталоге рабочего пространства (labs/lab02/report)

```
(rzemcov@rzemcov)-[~/.../study/2025-2026/Архитектура компьютера/study_arch-pc]
$ cd /home/rzemcov/work/study/2025-2026/"Архитектура компьютера"/study_arch-pc/labs/lab01/report/

(rzemcov@rzemcov)-[~/.../study_arch-pc/labs/lab01/report]
$ git status
```

Я перешел в каталог для сдачи лабораторной работы.

Задание №2. Скопируйте отчеты по выполнению предыдущих лабораторных работ в соответствующие каталоги созданного рабочего пространства.

```
(rzemcov@rzemcov)-[~/.../study_arch-pc/labs/lab01/report]
$ cp ~/Downloads/lab01_report.pdf report.md
```

В данном задании, я перенес лабораторную работу №1 в каталог, который меня просят

Задание №3. Загрузите файлы на github

```
(rzemcov@rzemcov)-[~/.../study_arch-pc/labs/lab01/report]
$ cp ~/Downloads/lab01_report.pdf report.md

(rzemcov@rzemcov)-[~/.../study_arch-pc/labs/lab01/report]
$ git add .
warning: in the working copy of 'labs/lab01/report/report.md', CRLF will be replaced by LF the next time Git
s it

(rzemcov@rzemcov)-[~/.../study_arch-pc/labs/lab01/report]
$ git commit -m 'feat(labs): add report for lab02 and copy lab01'
[master c12f06c] feat(labs): add report for lab02 and copy lab01
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 labs/lab01/report/lab01_report.pdf
create mode 100644 labs/lab01/report/report.md

(rzemcov@rzemcov)-[~/.../study_arch-pc/labs/lab01/report]
$ git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 1.39 MiB | 2.76 MiB/s, done.
Total 7 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 3 local objects.
To github.com:zemsovroman-a1ly/study_arch-pc.git
096b5e0..c12f06c master -> master

(rzemcov@rzemcov)-[~/.../study_arch-pc/labs/lab01/report]
$
```

Я выполнил самостоятельное задание. Создал файл для отчета по этой лабораторной работе и скопировал отчет по предыдущей. Все новые файлы я так же добавил, закоммитил и отправил на **GitHub**. Это закрепило мои практические навыки работы с

основными командами **Git**.

### **Выводы**

В ходе выполнения данной лабораторной работы я изучил теоретические основы и получил практические навыки работы с системой контроля версий Git. Я научился выполнять базовую настройку Git, создавать и настраивать репозитории на GitHub, использовать SSH-ключи для безопасного соединения, а также освоил основной рабочий цикл: добавление изменений (add), их фиксацию (commit) и отправку на удаленный сервер (push). Цель работы была полностью достигнута.

1. Введение - О системе контроля версий

[https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0\\_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F\\_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8)

2. What is Git version control? / <https://about.gitlab.com/topics/version-control/what-isgit-version-control/>

3. Основные команды GIT / <https://habr.com/ru/articles/918386/>