

# ***Отчёт по лабораторной работе***

***Лабораторная №2***

Полина Витальевна Барабаш

# ***Содержание***

<b><i>1</i></b>	<b><i>Цель работы</i></b>	<b><i>5</i></b>
<b><i>2</i></b>	<b><i>Задание</i></b>	<b><i>6</i></b>
<b><i>3</i></b>	<b><i>Выполнение лабораторной работы</i></b>	<b><i>7</i></b>
<b><i>4</i></b>	<b><i>Ответы на вопросы</i></b>	<b><i>16</i></b>
<b><i>5</i></b>	<b><i>Выводы</i></b>	<b><i>22</i></b>

## **Список иллюстраций**

3.1	Установка git . . . . .	7
3.2	Установка gh . . . . .	8
3.3	Задание имени и email репозитория . . . . .	8
3.4	Настройка utf-8, задание имени начальной ветки и параметров .	9
3.5	Создание ключа ssh по алгоритму rsa . . . . .	9
3.6	Создание ключа по алгоритму ed25519 . . . . .	10
3.7	Создание ключа PGP . . . . .	11
3.8	Вывод списка ключей PGP . . . . .	11
3.9	Вывод ключа по отпечатку . . . . .	12
3.10	Добавление ключа GPG на GitHub . . . . .	12
3.11	Настройка коммитов . . . . .	12
3.12	Успешная настройка gh . . . . .	13
3.13	Копирование шаблона рабочего пространства . . . . .	14
3.14	Удаление лишних файлов . . . . .	14
3.15	Создание необходимых каталогов . . . . .	15
3.16	Отправка файлов на сервер . . . . .	15

## ***Список таблиц***

4.1 Основные команды git . . . . .	18
------------------------------------	----

# ***1 Цель работы***

Целью данной работы является изучить идеологию и применение средств контроля версий и освоить умения по работе с git.

## ***2 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.

### ***3 Выполнение лабораторной работы***

#### **Задание 1. Установить git.**

Я вошла в режим суперпользователя и установила git с помощью команды `dnf install git` (рис. [3.1]).

```
[pvbarabash@pvbarabash ~]$ sudo -i
[sudo] пароль для pvbarabash:
[root@pvbarabash ~]# dnf install git
Последняя проверка окончания срока действия метаданных: 0:34:58 назад, Пт 01 мар 2024 13:47:57.
Пакет git-2.44.0-1.fc39.x86_64 уже установлен.
Зависимости разрешены.
Нет действий для выполнения.
Выполнено!
[root@pvbarabash ~]#
```

Рис. 3.1: Установка git

#### **Задание 2. Установить gh.**

Я установила gh с помощью команды `dnf install gh` (рис. [3.2]).

```
[root@pvbarabash ~]# dnf install gh
Последняя проверка окончания срока действия метаданных: 0ч37:04 назад, Пт 01 мар 2024 13:47:57.
Зависимости разрешены.
=====
Пакет                Архитектура          Версия
=====
Установка:
gh                    x86_64                2.43.1-1.fc39

Результат транзакции
=====
Установка 1 Пакет

Объем загрузки: 9.1 М
Объем изменений: 46 М
Продолжить? [д/Н]: y
Загрузка пакетов:
gh-2.43.1-1.fc39.x86_64.rpm
=====
Общий размер
Проверка транзакции
Проверка транзакции успешно завершена.
Идет проверка транзакции
Тест транзакции проведен успешно.
Выполнение транзакции
Подготовка          :
Установка           : gh-2.43.1-1.fc39.x86_64
Запуск скрипглета   : gh-2.43.1-1.fc39.x86_64
Проверка            : gh-2.43.1-1.fc39.x86_64

Установлен:
gh-2.43.1-1.fc39.x86_64

Выполнено!
```

Рис. 3.2: Установка gh

**Задание 3.** Произвести базовую настройку git. Задать имя и email владельца репозитория. Настроить utf-8 в выводе сообщений git. Задать имя начальной ветки, параметр autocrlf, параметр safecrlf.

Я задала имя и email своего репозитория с помощью двух команд:

```
git config --global user.name "Name Surname"
```

```
git config --global user.email "work@mail"
```

Вместо Name Surname я ввела имя своего репозитория и вместо work@mail почту, на которую зарегистрирована учетная запись (рис. [3.3]).

```
[root@pvbarabash ~]# git config --global user.name "pvbarabash"
[root@pvbarabash ~]# git config --global user.email "barabashpolina02@gmail.com"
[root@pvbarabash ~]#
```

Рис. 3.3: Задание имени и email репозитория

Я настроила utf-8 в выводе сообщений git с помощью команды git config --global core.quotepath false, задала имя начальной ветки master с помощью команды git config --global init.defaultBranch master, параметр autocrlf с помощью



git config --global core.autocrlf input и параметр safecrlf с помощью git config --global core.safecrlf warn (рис. [3.4]).

```
[root@pvbarabash ~]# git config --global core.quotepath false
[root@pvbarabash ~]# git config --global init.defaultBranch master
[root@pvbarabash ~]# git config --global core.autocrlf input
[root@pvbarabash ~]# git config --global core.safecrlf warn
[root@pvbarabash ~]#
```

Рис. 3.4: Настройка utf-8, задание имени начальной ветки и параметров

**Задание 4.** Создать ключи ssh по двум алгоритмам.

Я создала ключ ssh по алгоритму rsa с ключём размером 4096 бит с помощью команды ssh-keygen -t rsa -b 4096 (рис. [3.5]).

```
[root@pvbarabash ~]# ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:KnP2CGCMXI1PqtoRWroYGypo7bLUMMV8YtzUQ33vSwo root@pvbarabash
The key's randomart image is:
+---[RSA 4096]-----+
|      .O..      |
|    + =  o . .  |
|   X =  . . .   |
|.o+ *          o |
|.+* . . S  + .  |
|==o . . E o     |
|*oo.+ + .      |
|B0 o * o       |
|O.=. . .       |
+---[SHA256]-----+
[root@pvbarabash ~]#
```

Рис. 3.5: Создание ключа ssh по алгоритму rsa

Я создала ключ по алгоритму ed25519 с помощью команды ssh-keygen -t ed25519 (рис. [3.6]).

```

[root@pvbarabash ~]# ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_ed25519
Your public key has been saved in /root/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:ucshvypN7p+SVF95mvskMYeV/ip1I/x5/2GXZxQ3Jsg root@pvbarabash
The key's randomart image is:
+--[ED25519 256]--+
|
|      . . .
|      E oo+.
|      .. o++ +
|      .S. .+=o .
|      o .. o*.+o
|      =..o .o+=B
|      . += + oo,==
|      oo+B. oo *
+----[SHA256]-----+
[root@pvbarabash ~]#

```

Рис. 3.6: Создание ключа по алгоритму ed25519

### Задание 5. Создать ключи PGP.

Я сгенерировала ключ PGP с помощью команды `gpg --full-generate-key`. Из предложенных опций выбирала:

- тип RSA and RSA;
- размер 4096;
- срок действия 0 (срок действия не истекает никогда).

Я ввела личную информацию, которая сохранится в ключе: имя и адрес электронной почты (рис. [3.7]).

```
[root@pvbarabash ~]# 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: создан каталог '/root/.gnupg'
Выберите тип ключа:
  (1) RSA and RSA
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
  (9) ECC (sign and encrypt) *default*
 (10) ECC (только для подписи)
 (14) Existing key from card
Ваш выбор? 1
длина ключей RSA может быть от 1024 до 4096.
Какой размер ключа Вам необходим? (3072) 4096
Запрошенный размер ключа - 4096 бит
Выберите срок действия ключа.
  0 = не ограничен
  <n> = срок действия ключа - n дней
  <n>w = срок действия ключа - n недель
  <n>m = срок действия ключа - n месяцев
  <n>y = срок действия ключа - n лет
Срок действия ключа? (0) 0
Срок действия ключа не ограничен
Все верно? (y/N) y

GnuPG должен составить идентификатор пользователя для идентификации ключа.
Ваше полное имя: pvbarabash
Адрес электронной почты: barabashpolina02@gmail.com
Примечание:
Вы выбрали следующий идентификатор пользователя:
"pvbarabash <barabashpolina02@gmail.com>"
```

Рис. 3.7: Создание ключа PGP

### Задание 6. Добавить PGP ключ в GitHub.

Я вывела список ключей и скопировала отпечаток приватного ключа с помощью `gpg --list-secret-keys --keyid-format LONG`. Он начинается после `sec` (рис. [3.8]).

```
[root@pvbarabash ~]# gpg --list-secret-keys --keyid-format LONG
gpg: проверка таблицы доверия
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: глубина: 0 достоверных: 1 подписанных: 0 доверие: 0-, 0q, 0n, 0m, 0f, 1u
[keyboard]
-----
sec  rsa4096/3CA06DDA692372ED 2024-03-01 [SC]
     54A8D0F2F2F5F6E08E7083E83CA06DDA692372ED
uid          [ абсолютно ] pvbarabash <barabashpolina02@gmail.com>
ssb  rsa4096/57DE99E4FADAEB39 2024-03-01 [E]

[root@pvbarabash ~]#
```

Рис. 3.8: Вывод списка ключей PGP

Для копирования ключа у меня была ошибка с `xclip`, поэтому я вывела ключ с

помощью команды `gpg --armor --export` и скопировала его вручную (рис. [3.9]).

```
[root@pvbarabash ~]# gpg --armor --export PGP rsa4096/3CA06DDA692372ED 2024-03-01 [SC] 54A8D0F2
F2F5F6E08E7083E83CA06DDA692372ED
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGXhVNsBEACdoUMCT6MSenHMgz/4KG7I22yKDagyR1ULoBqeyxAlmgoICTM
dzjEg4/r+MFG6ab2aubraQKRSK1KqKmcITEN0s52jnEuD+rLKmmasbNESCYSAFSx
QbBX1Scr+JC8rNBZzpkv8k6FREgp9albeBHTy1UpSrhwi1BrH9+8a3jdjMXGqLSt
zGQ144zMvUwKBzB9n8iUML9fE58f7u8szNIEJPFr4H+rLDcs1hVPpQJ83FcRtot
LUxXg44smW/evvXpmMoWQqrEZ9UAsDt3JhCMhIMvczH9LfB48EPFHeBYrCfZ8r5X
```

Рис. 3.9: Вывод ключа по отпечатку

Я перешла в настройки GitHub, нажала на кнопку New GPG key и вставила полученный ключ в поле ввода (рис. [3.10]).

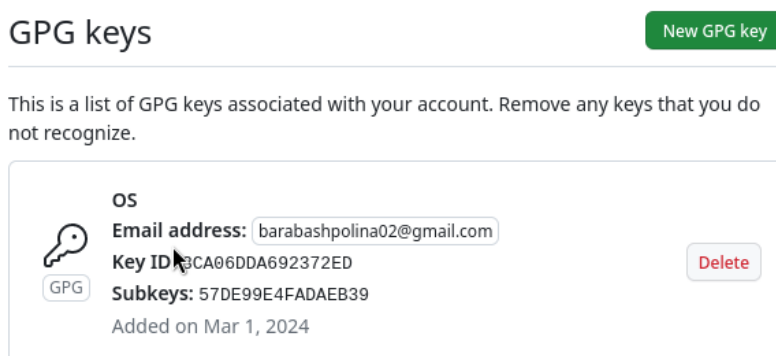


Рис. 3.10: Добавление ключа GPG на GitHub

**Задание 7.** Используя введённый email, указать Git применять его при подписи КОММИТОВ

Я указала Git применять email при подписи коммитов с помощью следующих команд:

```
git config --global user.signingkey
```

```
git config --global commit.gpgsign true
```

```
git config --global gpg.program $(which gpg2) (рис. [3.11]).
```

```
[root@pvbarabash os-intro]# git config --global commit.gpgsign true
[root@pvbarabash os-intro]# git config --global gpg.program $(which gpg2)
[root@pvbarabash os-intro]#
```

Рис. 3.11: Настройка коммитов

### Задание 8. Настроить gh.

Сначала я авторизовалась с помощью команды `gh auth login` и ответила на наводящие вопросы, а затем авторизовалась через браузер (рис. [3.12]).

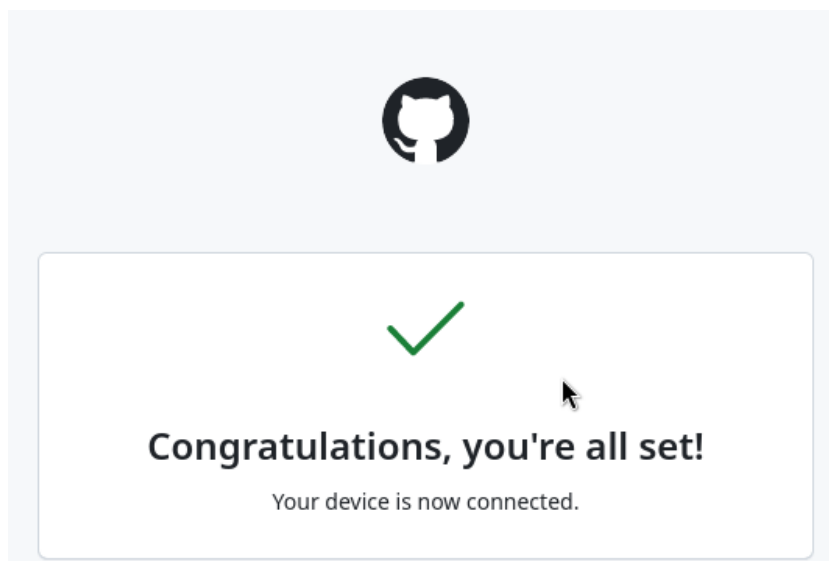


Рис. 3.12: Успешная настройка gh

### Задание 9. Создать шаблон рабочего пространства.

Я создала нужный каталог с помощью команды `mkdir -p ~/work/study/2022-2023/“Операционные системы”`. Перешла в него с помощью команды `cd`. Так как на github у меня уже был скопирован шаблон рабочего пространства, я клонировала его с помощью команды `git clone -recursive git@github.com:/study_2022-2023_os-intro.git os-intro` (рис. [3.13]).

```
[root@pvbarabash Операционные системы]# git clone --recursive git@github.com:pvbarabash/study_2022-2023_os-intro.git os-intro
Клонирование в «os-intro»...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 32 (delta 1), reused 18 (delta 0), pack-reused 0
Получение объектов: 100% (32/32), 18.60 КиБ | 2.32 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 «/root/work/study/2023-2024/Операционные системы/os-intro/template/presentation»...
remote: Enumerating objects: 95, done.
remote: Counting objects: 100% (95/95), done.
remote: Compressing objects: 100% (67/67), done.
remote: Total 95 (delta 34), reused 87 (delta 26), pack-reused 0
Получение объектов: 100% (95/95), 96.99 КиБ | 841.00 КиБ/с, готово.
Определение изменений: 100% (34/34), готово.
Клонирование в «/root/work/study/2023-2024/Операционные системы/os-intro/template/report»...
remote: Enumerating objects: 126, done.
remote: Counting objects: 100% (126/126), done.
remote: Compressing objects: 100% (87/87), done.
remote: Total 126 (delta 52), reused 108 (delta 34), pack-reused 0
Получение объектов: 100% (126/126), 335.80 КиБ | 1.87 МиБ/с, готово.
Определение изменений: 100% (52/52), готово.
Submodule path 'template/presentation': checked out '40a1761813e197d00e8443ff1ca72c60a304f24c'
Submodule path 'template/report': checked out '7c31ab8e5dfa8cdb2d67caeb8a19ef8028ced88e'
[root@pvbarabash Операционные системы]#
```

Рис. 3.13: Копирование шаблона рабочего пространства

#### Задание 10. Настроить каталог курса.

Я перешла в каталог курса с помощью команды `cd ~/work/study/2022-2023/“Операционные системы”/os-intro` и удалила лишние файлы с помощью команды `rm package.json` (рис. [3.14]).

```
[root@pvbarabash os-intro]# rm package.json
rm: удалить обычный файл 'package.json'? yes
[root@pvbarabash os-intro]#
```

Рис. 3.14: Удаление лишних файлов

Я создала необходимые каталоги с помощью команд `echo os-intro > COURSE` и `make` (рис. [3.15]).

```
[root@pvbarabash os-intro]# echo os-intro > COURSE
[root@pvbarabash os-intro]# make
Usage:
  make <target>

Targets:
  list           List of courses
  prepare       Generate directories structure
  submodule      Update submules
[root@pvbarabash os-intro]#
```

Рис. 3.15: Создание необходимых каталогов

Затем я отправила файлы на сервер (рис. [3.16]).

```
[root@pvbarabash os-intro]# git push
Everything up-to-date
[root@pvbarabash os-intro]#
```

Рис. 3.16: Отправка файлов на сервер

## **4 Ответы на вопросы**

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

Система контроля версий это — программное обеспечение для облегчения работы с изменяющейся информацией. Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом.

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

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. Также он может добавлять коммиты — комментарии о том, что было изменено. Пользователь создает рабочую копию. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент — система хранит историю версий.

- 3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.**



В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. То есть классические системы контроля версий являются централизованными. В отличие от классических, в децентрализованных 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. Опишите действия с VCS при единоличной работе с хранилищем.**

При единоличной работе с VCS доступны те же преимущества, что и при совместном использовании (сохранение, откат версий). Отличие в том, что работа с хранилищем доступна только одному пользователю.

#### **5. Опишите порядок работы с общим хранилищем VCS.**

Для удобства совместной работы с общим хранилищем VCS, стоит добавлять комментарии, что было изменено и подписывать кем были сделаны изменения.

#### **6. Каковы основные задачи, решаемые инструментальным средством git?**

Основными задачами, решаемыми инструментальным средством git, являются задачи не потерять файлы с исходным кодом, защититься от случайных исправлений и удалений, отменить изменения, если они оказались некорректными, одновременно поддерживать рабочую версию и разработку новой.

#### **7. Назовите и дайте краткую характеристику командам git.**

Таблица 4.1: Основные команды git

Команда	Значение
git init	Создание основного дерева репозитория
git pull	Получение обновлений (изменений) текущего дерева из центрального репозитория
git push	Отправка всех произведённых изменений локального дерева в центральный репозиторий
git status	Просмотр списка изменённых файлов в текущей директории
git diff	Добавить все изменённые и/или созданные файлы и/или каталоги
git add .	Добавить все изменённые и/или созданные файлы и/или каталоги
git add имена_файлов	Добавить конкретные изменённые и/или созданные файлы и/или каталоги
git rm имена_файлов	Удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории)
git commit -am 'Описание коммита'	Сохранить все добавленные изменения и все изменённые файлы
git commit	Сохранить добавленные изменения с внесением комментария через встроенный редактор

Команда	Значение
<code>git checkout -b имя_ветки</code>	Создание новой ветки, базирующейся на текущей
<code>git checkout имя_ветки</code>	Переключение на некоторую ветку
<code>git push origin имя_ветки</code>	Отправка изменений конкретной ветки в центральный репозиторий
<code>git merge --no-ff имя_ветки</code>	Слияние ветки с текущим деревом
<code>git branch -d имя_ветки</code>	Удаление локальной уже слитой с основным деревом ветки
<code>git branch -D имя_ветки</code>	Принудительное удаление локальной ветки
<code>git push origin :имя_ветки</code>	Удаление ветки с центрального репозитория

## 8. Приведите примеры использования при работе с локальным и удалённым репозиториями.

Пример использования при работе с удалённым репозиторием:

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
```

Пример использования при работе с локальным репозиторием:

Для инициализации локального репозитория, расположенного, например, в каталоге `~/tutorial`, необходимо ввести в командной строке:

```
cd ~
```

```
mkdir tutorial
```

cd tutorial

git init

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

Под веткой принято понимать независимую последовательность коммитов в хронологическом порядке. Однако конкретно в Git реализация ветки выполнена как указатель на последний коммит в рассматриваемой ветке. После создания ветки уже новый указатель ссылается на текущий коммит.

Имя основной ветки Git-проекта по умолчанию — master (однако зачастую бывает main, например, в GitHub), она появляется сразу при инициализации репозитория. Эта ветка ничем не отличается от остальных и также ее можно переименовать, но по договоренности master принято считать главной веткой в проекте.

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

В процессе работы над любым проектом в директории с кодом создаются файлы, которые не являются частью исходного кода. Все эти файлы можно условно разделить на несколько групп:

- Инструментарий:
  - служебные файлы
  - конфигурационные файлы
  - временные файлы редакторов
- Временные файлы:
  - логи — в них содержится полезная информация для отладки, которая собирается во время запуска и работы приложения
  - кеши — файлы, которые нужны для ускорения разных процессов
- Артефакты:

- результаты сборки проекта — например, после компиляции или сборки фронтенда
- зависимости, которые устанавливаются во время разработки — например, `node_modules` или `vendor`
- результаты выполнения тестов — например, информация о покрытии кода тестами

Все это в обычной ситуации не должно попадать в репозиторий.

Git позволяет гибко настраивать игнорирование определенных файлов и директорий. Делается это с помощью файла `.gitignore`, который нужно создать в корне проекта. В этот файл с помощью текстового редактора добавляются имена файлов и директорий, которые надо игнорировать.

## ***5 Выводы***

При выполнении данной лабораторной работы я изучила идеологию и применение средств контроля версий и освоила умения по работе с git.