

# **Отчёт по лабораторной работе №2**

**Операционные системы**

Мурашов Иван Вячеславович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
4.1	Установка программного обеспечения . . . . .	8
4.2	Базовая настройка git . . . . .	8
4.3	Создание ключей . . . . .	9
4.4	Настройка github и добавление ключей . . . . .	10
4.5	Настройка автоматических подписей коммитов git и авторизация	12
4.6	Создание репозитория курса на основе шаблона и настройка каталога . . . . .	13
<b>5</b>	<b>Ответы на контрольные вопросы</b>	<b>15</b>
<b>6</b>	<b>Выводы</b>	<b>19</b>

## Список иллюстраций

4.1	Установка git . . . . .	8
4.2	Установка gh . . . . .	8
4.3	Задание имени и email . . . . .	8
4.4	Настройка utf-8 . . . . .	9
4.5	Настройка autocrlf и safecrlf . . . . .	9
4.6	Создание ssh ключа . . . . .	9
4.7	Создание ssh ключа . . . . .	9
4.8	Создание pgr ключа . . . . .	10
4.9	Профиль github . . . . .	11
4.10	Список ключей . . . . .	12
4.11	Копирование pgr ключа . . . . .	12
4.12	Список ключей . . . . .	12
4.13	Настройка автоматических подписей . . . . .	13
4.14	Авторизация . . . . .	13
4.15	Создание репозитория . . . . .	13
4.16	Клонирование репозитория . . . . .	14
4.17	Настройка каталога курса . . . . .	14
4.18	Отправка файлов на сервер . . . . .	14

## Список таблиц

# 1 Цель работы

Целью данной лабораторной работы является изучение идеологии, применение средств контроля версий и освоение умений по работе с git.

## 2 Задание

1. Установить программное обеспечение
2. Создать базовую конфигурацию для работы с git
3. Создать ключ SSH
4. Создать ключ PGP
5. Создать профиль на GitHub и авторизоваться на устройстве
6. Создать репозиторий курса на основе шаблона
7. Настроить каталог курса

### 3 Теоретическое введение

Целью данной лабораторной работы является изучение идеологии, применение средств контроля версий и освоение умений по работе с git. 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.

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 Установка программного обеспечения

Для начала я устанавливаю git (рис. 4.1).

```
root@ivmurashov:~# dnf install git
Последняя проверка окончания срока действия метаданных: 1:12:30 назад, Пн 19 ф
ев 2024 15:41:15.
Пакет git-2.43.0-1.fc39.x86_64 уже установлен.
Зависимости разрешены.
Нет действий для выполнения.
Выполнено!
```

Рис. 4.1: Установка git

Затем устанавливаю gh (рис. 4.2).

```
root@ivmurashov:~# dnf install gh
Последняя проверка окончания срока действия метаданных: 1:13:23 назад, Пн 19 ф
ев 2024 15:41:15.
Пакет gh-2.43.1-1.fc39.x86_64 уже установлен.
Зависимости разрешены.
Нет действий для выполнения.
Выполнено!
```

Рис. 4.2: Установка gh

### 4.2 Базовая настройка git

Задаю имя и email владельца репозитория (рис. 4.3).

```
root@ivmurashov:~# git config --global user.name "neve7mind"
root@ivmurashov:~# git config --global user.email "ivan.murashov1200@gmail.com"
```

Рис. 4.3: Задание имени и email



Настраиваю utf-8 в выводе сообщений git и задаю имя начальной ветки (рис. 4.4).

```
root@ivmurashov:~# git config --global core.quotePath false
root@ivmurashov:~# git config --global init.defaultBranch master
```

Рис. 4.4: Настройка utf-8

Настраиваю параметры autocrlf и safecrlf (рис. 4.5).

```
error: key does not contain a section: core.autocrlf
root@ivmurashov:~# git config --global core.autocrlf input
root@ivmurashov:~# git config --global core.safecrlf warn
```

Рис. 4.5: Настройка autocrlf и safecrlf

## 4.3 Создание ключей

Создаю ключ SSH по алгоритму rsa с ключом размером 4096 бит (рис. 4.6).

```
root@ivmurashov:~# ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
```

Рис. 4.6: Создание ssh ключа

А затем по алгоритму ed25519 (рис. 4.7).

```
root@ivmurashov:~# ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
```

Рис. 4.7: Создание ssh ключа

Генерирую ключ PGP, выбирая из предложенных опций тип 'RSA and RSA', размер 4096, без срока действия. Указываю имя и адрес email (рис. 4.8).

```

root@ivmurashov:~# 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.

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)
Срок действия ключа не ограничен
Все верно? (y/N) y

GnuPG должен составить идентификатор пользователя для идентификации ключа.

```

Рис. 4.8: Создание pgp ключа

## 4.4 Настройка github и добавление ключей

У меня уже был создан профиль на github (рис. 4.9).



**ivmurashov**

neve7mind

Edit profile

👤 1 follower · 1 following

Рис. 4.9: Профиль github

Вывожу список ключей (рис. 4.10).

```

root@ivmurashov:~# 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   rsa4096/14A44B0AAB5809A0 2024-02-19 [SC]
      51EDBD9F85C58DBEF512FB1314A44B0AAB5809A0
uid           [ абсолютно ] neve7mind <ivan.murashov1200@gmail.com>
ssb   rsa4096/26D5F9A2513E79DF 2024-02-19 [E]

```

Рис. 4.10: Список ключей

Копирую сгенерированный pgr ключ в буфер обмена (рис. 4.11).

```

root@ivmurashov:~# gpg --armor --export 14A44B0AAB5809A0 | xclip -sel clip

```

Рис. 4.11: Копирование pgr ключа

Перехожу в настройки GitHub, нажимаю на кнопку ‘New GPG key’ и вставляю скопированный ключ в поле ввода (рис. 4.12).

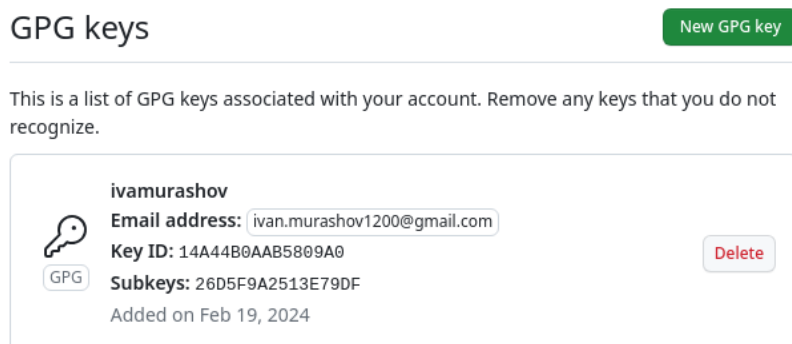


Рис. 4.12: Список ключей

## 4.5 Настройка автоматических подписей коммитов git и авторизация

Используя введенный email, указываю Git применять его при подписи коммитов (рис. 4.13).

```

ivmurashov@ivmurashov:~$ git config --global user.signingkey 14A44B0AAB580
ivmurashov@ivmurashov:~$ git config --global commit.gpgsign true
ivmurashov@ivmurashov:~$ pg2 --list-keys --keyid-format LONG
bash: pg2: команда не найдена...
ivmurashov@ivmurashov:~$ gpg2 --list-keys --keyid-format LONG
[keyboard]
-----
pub      rsa4096/F03F7ACF8D648D18 2024-02-19 [SC]
         E1C02067A0966AE1DDF46793F03F7ACF8D648D18
uid            [ абсолютно ] neve7mind <ivan.murashov1200@gmail.com>
sub      rsa4096/6DF0A6C2D0937322 2024-02-19 [E]

ivmurashov@ivmurashov:~$ git config --global gpg.program F03F7ACF8D648D18

```

Рис. 4.13: Настройка автоматических подписей

Авторизуюсь с помощью браузера (gh login) (рис. 4.14).

```

ivmurashov@ivmurashov:~$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations on this host? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: 2AAB-01EF
Press Enter to open github.com in your browser...
Окно или вкладка откроются в текущем сеансе браузера.
✓ Authentication complete.
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as neve7mind

```

Рис. 4.14: Авторизация

## 4.6 Создание репозитория курса на основе шаблона и настройка каталога

Создаю каталог курса, перемещаюсь в него и создаю репозиторий на основе шаблона (рис. 4.15).

```

ivmurashov@ivmurashov:~$ mkdir -p ~/work/study/2023-2024/"Операционные системы"
ivmurashov@ivmurashov:~$ cd ~/work/study/2023-2024/"Операционные системы"
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы$ gh repo create study_2023-2024_os-intro --template=yamadharma/course-directory-student-template --public
✓ Created repository neve7mind/study_2023-2024_os-intro on GitHub
https://github.com/neve7mind/study_2023-2024_os-intro

```

Рис. 4.15: Создание репозитория

Клонирую репозиторий (рис. 4.16).

```
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы$ git clone --recursive
git@github.com:neve7mind/study_2023-2024_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.
```

Рис. 4.16: Клонирование репозитория

Удаляю лишние файлы, создаю необходимые каталоги (рис. 4.17).

```
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ rm package.js
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ echo os-intro > COURSE
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ make
Usage:
  make <target>

Targets:
  list           List of courses
  prepare        Generate directories structure
  submodule      Update submules

ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ make prepare
```

Рис. 4.17: Настройка каталога курса

Отправляю файлы на сервер (рис. 4.18).

```
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ git
add .
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ git
commit -am 'feat(main): make course structure'
Текущая ветка: master
Эта ветка соответствует «origin/master».

ничего коммитить, нет изменений в рабочем каталоге
ivmurashov@ivmurashov:~/work/study/2023-2024/Операционные системы/os-intro$ git
push
Everything up-to-date
```

Рис. 4.18: Отправка файлов на сервер

## 5 Ответы на контрольные вопросы

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

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.

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

Хранилище - это репозиторий, в котором хранятся все документы, включая историю изменений.

Commit - отслеживание и сохранение изменений.

История - сохраняет в себе изменения проекта на всех его этапах.

Рабочая копия - копия проекта на основе версии из хранилища.

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

Централизованные VCS (CVS, SVN, TFS) - одно главное хранилище всего проекта, каждый участник проекта добавляет изменения в данное хранилище.

Децентрализованные VCS (Git, Bazaar) - у каждого участника свой вариант репозитория, что позволяет соединять впоследствии нужные ветки проекта.

4. Опишите действия с VCS при единоличной работе с хранилищем.

Создаётся и подключается удалённый репозиторий. Участник проекта при взаимодействии с данным репозиторием отправляет все изменения на сервер.

5. Опишите порядок работы с общим хранилищем VCS.

Участник получает определённую версию проекта в хранилище. Участник отправляет все изменения на сервер. При этом старые версии проекта сохраняются.

6. Каковы основные задачи, решаемые инструментальным средством git?

Хранение информации об изменениях проекта, возможность совместной работы над одним проектом.

7. Назовите и дайте краткую характеристику командам git.

Создание основного дерева репозитория: `git init`

Получение обновлений (изменений) текущего дерева из центрального репозитория: `git pull`

Отправка всех произведённых изменений локального дерева в центральный репозиторий: `git push`

Просмотр списка изменённых файлов в текущей директории: `git status`

Просмотр текущих изменений: `git diff`

Сохранение текущих изменений: добавить все изменённые и/или созданные файлы и/или каталоги: `git add .`

добавить конкретные изменённые и/или созданные файлы и/или каталоги: `git add <имена_>`

удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в директории): `git rm <имена_>`



### Сохранение добавленных изменений:

сохранить все добавленные изменения и все изменённые файлы: `git commit -am 'Описание коммита'`

сохранить добавленные изменения с внесением комментария через встроенный редактор: `git commit`

создание новой ветки, базирующейся на текущей: `git checkout -b имя_ветки`

переключение на некоторую ветку: `git checkout имя_ветки`

отправка изменений конкретной ветки в центральный репозиторий: `git push origin имя_ветки`

слияние ветки с текущим деревом: `git merge --no-ff имя_ветки`

### Удаление ветки:

удаление локальной уже слитой с основным деревом ветки: `git branch -d имя_ветки`

принудительное удаление локальной ветки: `git branch -D имя_ветки`

удаление ветки с центрального репозитория: `git push origin :имя_ветки`

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

`git commit -am 'feat(main): make course structure'` - сохранение всех добавленных изменений и всех изменённых файлов

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

Ветви - это варианты развития проекта в хранилище. Они нужны для удобства коллективной работы с хранилищем, а так же для создания “пробных” версий проекта

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

Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл .gitignore с помощью сервисов. Для этого сначала нужно получить список имеющихся шаблонов:

```
curl -L -s https://www.gitignore.io/api/list
```

Затем скачать шаблон, например, для C и C++

```
curl -L -s https://www.gitignore.io/api/c » .gitignore curl -L -s https://www.gitignore.io/api/c++  
» .gitignore
```

## 6 Выводы

В ходе выполнения данной лабораторной работы я изучил идеологии и применение средств контроля версий и освоил умения по работе с git.