

# **Отчёт по лабораторной работе №2**

**Дисциплина: Операционные системы**

**Шошина Евгения Александровна, группа: НКАбд-03-22**

# 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>
3.0.1	Системы контроля версий. Общие понятия . . . . .	7
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.0.1	Установка программного обеспечения . . . . .	9
4.0.2	Создали ключи ssh . . . . .	9
4.0.3	Создали ключи pgr . . . . .	10
4.0.4	Настройка github . . . . .	11
4.0.5	Добавили PGP ключ в GitHub . . . . .	11
4.0.6	Настройка автоматических подписей коммитов git . . . . .	12
4.0.7	Настройка gh . . . . .	12
4.0.8	Создание репозитория курса на основе шаблона . . . . .	12
4.0.9	Настройка каталога курса . . . . .	13
<b>5</b>	<b>Выводы</b>	<b>14</b>
<b>6</b>	<b>Контрольные вопросы</b>	<b>15</b>

## **Список иллюстраций**

## **Список таблиц**

# 1 Цель работы

- 1) Изучить идеологию и применение средств контроля версий.
- 2) Освоить умения по работе с git.

## 2 Задание

- 1) Создать базовую конфигурацию для работы с git.
- 2) Создать ключ SSH.
- 3) Создать ключ PGP.
- 4) Настроить подписи git.
- 5) Зарегистрироваться на Github.
- 6) Создать локальный каталог для выполнения заданий по предмету.

## 3 Теоретическое введение

### 3.0.1 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.



## 4 Выполнение лабораторной работы

### 4.0.1 Установка программного обеспечения

Установка git - `dnf install git` Установка gh - `dnf install gh` `[eashoshina@fedora ~]$ dnf install gh`

### Базовая настройка git Зададим имя и email владельца репозитория:

```
[eashoshina@fedora ~]$ git config --global user.name "EAShoshina"
[eashoshina@fedora ~]$ git config --global user.email "lady.shoshina2017@yandex.ru"
```

Настроим utf-8 в выводе сообщений git: `[eashoshina@fedora ~]$ git config --global core.quotepath fa`

Настройте верификацию и подписание коммитов git (см. Верификация коммитов git с помощью GPG). Зададим имя начальной ветки (будем называть её master): `[eashoshina@fedora ~]$ git config --global init.defaultBranch master`

Параметр autocrlf: `[eashoshina@fedora ~]$ git config --global core.autocrlf input`

Параметр safecrlf: `[eashoshina@fedora ~]$ git config --global core.safecrlf warn`

### 4.0.2 Создали ключи ssh

по алгоритму rsa с ключом размером 4096 бит(`ssh-keygen -t rsa -b 4096`) по алгоритму ed25519(`ssh-keygen -t ed25519`)

### 4.0.3 Создали ключи pgr

```
[eashoshina@fedora ~]$ gpg --full-generate-key
gpg (GnuPG) 2.3.7; Copyright (C) 2021 Free Software Foundation, Inc.
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) 0
Срок действия ключа не ограничен
Все верно? (y/N) y

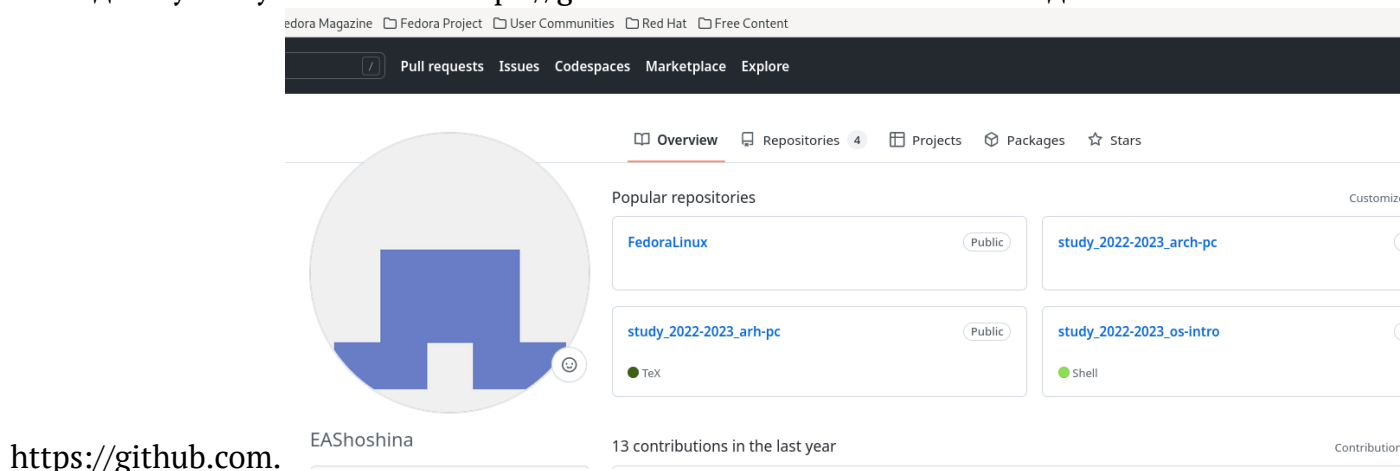
GnuPG должен составить идентификатор пользователя для идентификации ключа.

Ваше полное имя: EAShoshina
Адрес электронной почты: lady.shoshina2017@yandex.ru
Примечание:
Вы выбрали следующий идентификатор пользователя:
  "EAShoshina <lady.shoshina2017@yandex.ru>"
```

Генерируем ключ

## 4.0.4 Настройка github

Создали учётную запись на <https://github.com>. Заполнили основные данные на



## 4.0.5 Добавили PGP ключ в GitHub

Выводим список ключей и копируем отпечаток приватного ключа: Скопирован-




ли наш сгенерированный PGP ключ в буфер обмена: [eashoshina@fedora ~]\$ gpg --armor --export 566A34

Перешли в настройки GitHub (<https://github.com/settings/keys>), нажали на кнопку New GPG key и вставили полученный ключ в поле ввода.

## GPG keys

[New GPG key](#)

This is a list of GPG keys associated with your account. Remove any keys that you do not recognize.



**GPG**

**Email address:** lady.shoshina2017@yandex.ru

**Key ID:** 566A34F1A4E3FF00

**Subkeys:** 27F35FAD2030F3CA

Added on Feb 24, 2023

[Delete](#)

### 4.0.6 Настройка автоматических подписей коммитов git

Используя введённый email, указали Git, чтобы применять при подписи комми-

```
[eashoshina@fedora ~]$ git config --global user.signingkey 566A34F1A4E3FF00  
[eashoshina@fedora ~]$ git config --global commit.gpgsign true  
ТОБ: [eashoshina@fedora ~]$ git config --global gpg.program $(which gpg2)
```

### 4.0.7 Настройка gh

```
[eashoshina@fedora ~]$ gh auth login  
? What account do you want to log into? GitHub.com  
  
? You're already logged into github.com. Do you want to re-authenticate?  
  
? What is your preferred protocol for Git operations? SSH  
  
? Upload your SSH public key to your GitHub account? /home/eashoshina/.ssh  
  
? Title for your SSH key: 24 february  
  
? How would you like to authenticate GitHub CLI? Login with a web browser  
  
! First copy your one-time code: 28DF-57A9  
  
Press Enter to open github.com in your browser...
```

Для начала авторизовались

### 4.0.8 Создание репозитория курса на основе шаблона

Необходимо создать шаблон рабочего пространства (см. Рабочее пространство для лабораторной работы).

Например, для 2022–2023 учебного года и предмета «Операционные системы» (код предмета os-intro) создание репозитория примет следующий вид:

```
[eashoshina@fedora ~]$ mkdir -p ~/work/study/2022-2023/"Операционные системы"
[eashoshina@fedora ~]$ cd ~/work/study/2022-2023/"Операционные системы"
[eashoshina@fedora Операционные системы]$ gh repo create study_2022-2023_os-intro --template=yamadharma/course-directory-student-template --public
✓ Created repository EAShoshina/study_2022-2023_os-intro on GitHub
[eashoshina@fedora Операционные системы]$ git clone --recursive git@github.com:EAShoshina/study_2022-2023_os-intro.git os-intro
Клонирование в «os-intro»...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 27 (delta 1), reused 11 (delta 0), pack-reused 0
Получение объектов: 100% (27/27), 16.93 КиБ | 8.47 МиБ/с, готово.
Определение изменений: 100% (1/1), готово.
```

## 4.0.9 Настройка каталога курса

Перешли в каталог курса: Удалили лишние файлы: Создайте необходимые каталоги:

```
[eashoshina@fedora os-intro]$ ls
CHANGELOG.md  COURSE  LICENSE  prepare  project-personal  README.git-flow.md  template
LOGI: config    labs    Makefile  presentation  README.en.md      README.md
```

```
[eashoshina@fedora os-intro]$ git add .
[eashoshina@fedora os-intro]$ git commit -am 'feat(main): make course'
error: cannot run 676868D4B8DCC44F: Нет такого файла или каталога
error: gpg не удалось подписать данные
fatal: сбой записи объекта коммита
```

Отправьте файлы на сервер:

## 5 Выводы

- 1) Изучили идеологию и применение средств контроля версий.
- 2) Освоили умения по работе с git.

## 6 Контрольные вопросы

### 6.0.0.1 1) Система контроля версий (Version Control System, VCS)

— программное обеспечение для облегчения работы с изменяющейся информацией. ##### Система контроля версий (Version Control System, VCS) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. VCS 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. ##### 2) Хранилище (repository), или репозиторий, — место хранения файлов и их версий, служебной информации. ##### Commit («[трудовой] вклад», не переводится) — процесс создания новой версии; иногда синоним версии. ##### Версия (revision), или ревизия, — состояние всего хранилища или отдельных файлов в момент времени («пункт истории»). ##### Рабочая копия (working copy) — текущее состояние файлов проекта (любой версии), полученных из хранилища и, возможно, изменённых.

Хранилище – репозиторий - место хранения всех версий и служебной информации. Commit — это команда для записи индексированных изменений в репозиторий. История – место, где сохраняются все коммиты, по которым можно посмотреть данные о коммитах. Рабочая копия – текущее состояние файлов проекта, основанное на версии, загруженной из хранилища. ##### 3) Централизованные VCS: Одно основное хранилище всего проекта. Каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет и, затем, добавляет свои изменения обратно (Subversion, CVS, TFS, VAULT, AccuRev) #####

Децентрализованные VCS: У каждого пользователя свой вариант (возможно не один) репозитория. Присутствует возможность добавлять и забирать изменения из любого репозитория (Git, Mercurial, Bazaar) ##### 4) Опишите действия с VCS при единоличной работе с хранилищем. В рабочей копии, которую исправляет человек, появляются правки, которые отправляются в хранилище на каждом из этапов. То есть в правки в рабочей копии появляются, только если человек делает их (отправляет их на сервер) и никак по-другому. ##### 5) Опишите порядок работы с общим хранилищем VCS. Если хранилище общее, то в рабочую копию каждого, кто работает над проектом, приходят изменения, отправленные на сервер одним из команды. Рабочая правка каждого может изменяться вне зависимости от того, делает ли конкретный человек правки или нет. ##### 6) Каковы основные задачи, решаемые инструментальным средством git? У Git две основных задачи: первая — хранить информацию обо всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом. ##### 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: `git add` . – добавить все изменённые и/или созданные файлы и/или каталоги: `git add` – добавить конкретные изменённые и/или созданные файлы и/или каталоги: `git add` – удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): `git rm` имена\_файлов – 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 (при переключении на ветку, которой ещё нет в локальном репозитории).



рии, она будет создана и связана с удалённой) – отправка изменений конкретной ветки в центральный репозиторий: `git push origin имя_ветки` – слияние ветки с текущим деревом: `git merge --no-ff имя_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. Работа с удаленным репозиторием: `git remote` – просмотр списка настроенных удаленных репозиториев. Работа с локальным репозиторием: `git status` - выводит информацию обо всех изменениях, внесенных в дерево директорий проекта по сравнению с последним коммитом рабочей ветки. ##### 9) Что такое и зачем могут быть нужны ветви (branches)? Ветка (англ. branch) — это последовательность коммитов, в которой ведётся параллельная разработка какого-либо функционала. Ветки нужны, чтобы несколько программистов могли вести работу над одним и тем же проектом или даже файлом одновременно, при этом не мешая друг другу. Кроме того, ветки используются для тестирования экспериментальных функций: чтобы не повредить основному проекту, создается новая ветка специально для экспериментов. ##### 10) Как и зачем можно игнорировать некоторые файлы при commit? Игнорируемые файлы — это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты. В Git нет специальной команды для указания игнорируемых файлов: вместо этого необходимо вручную отредактировать файл . Временно игнорировать изменения в файле можно командой `git update-index --assumeunchanged`. # Список литературы{unnumbered}