

Отчет по лабораторной работе №2

дисциплина: Операционные системы

Старков Никита Алексеевич

Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	6
3.1	Базовая настройка git	6
3.2	Создание ключей и добавление их на гитхаб	7
3.3	Создание рабочей среды	9
4	Выводы	14

Список иллюстраций

3.1	Базовая настройка git	6
3.2	Настройка utf-8	6
3.3	Настройка верификации	6
3.4	Настройка верификации	6
3.5	Создание ssh ключа по алгоритму rsa	7
3.6	Создание ssh ключа по алгоритму ed25519	7
3.7	Создание ключа PGP	7
3.8	Добавление ключа	7
3.9	Копирование ключа в буфер обмена	7
3.10	Добавление ключа на сайте	8
3.11	Автоматические подписи git	8
3.12	Добавление ssh ключа	8
3.13	Создание каталога	8
3.14	Клонирование рабочей среды	9
3.15	Настройка каталога курса	9

1 Цель работы

Цель работы: изучить идеологию и применение средств контроля версий, освоить умения работы с git

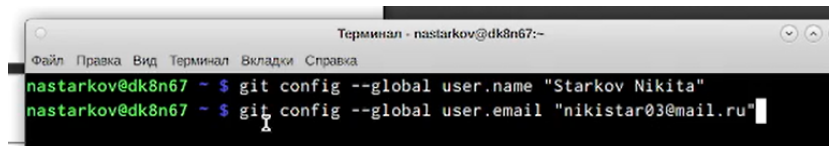
2 Задание

- 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

3.1 Базовая настройка git

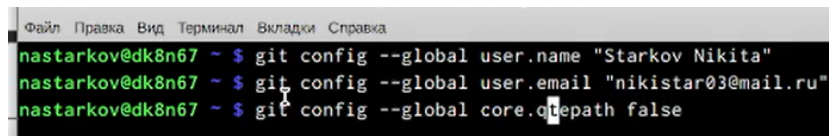
1) Вводим команды для базовой настройки git (задаем имя и email)



```
Терминал - nastarkov@dk8n67:~  
Файл Правка Вид Терминал Вкладки Справка  
nastarkov@dk8n67 ~ $ git config --global user.name "Starkov Nikita"  
nastarkov@dk8n67 ~ $ git config --global user.email "nikistar03@mail.ru"
```

Рис. 3.1: Базовая настройка git

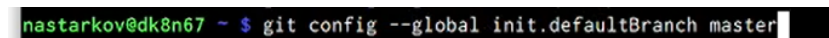
Настраиваем utf-8 для вывода сообщений в git



```
Файл Правка Вид Терминал Вкладки Справка  
nastarkov@dk8n67 ~ $ git config --global user.name "Starkov Nikita"  
nastarkov@dk8n67 ~ $ git config --global user.email "nikistar03@mail.ru"  
nastarkov@dk8n67 ~ $ git config --global core.editor false
```

Рис. 3.2: Настройка utf-8

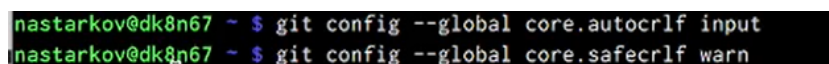
Настраиваем верификацию и подписание коммитов git и задаем имя начальной ветки



```
nastarkov@dk8n67 ~ $ git config --global init.defaultBranch master
```

Рис. 3.3: Настройка верификации

Далее прописываем параметры autocrlf и safecrlf



```
nastarkov@dk8n67 ~ $ git config --global core.autocrlf input  
nastarkov@dk8n67 ~ $ git config --global core.safecrlf warn
```

Рис. 3.4: Настройка верификации

3.2 Создание ключей и добавление их на гитхаб

2) Создаем ключи ssh (по алгоритму rsa с ключом размером 4096 бит и по алгоритму ed25519)

```
nastarkov@dk8n67 ~ $ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/.ssh/id_rsa):
```

Рис. 3.5: Создание ssh ключа по алгоритму rsa

```
nastarkov@dk8n67 ~ $ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/.ssh/id_ed25519):
```

Рис. 3.6: Создание ssh ключа по алгоритму ed25519

3) Генерируем ключ pgr и выбираем предложенные опции в соответствии с лабораторной работой

```
nastarkov@dk8n67 ~ $ gpg --full-generate-key
gpg (GnuPG) 2.2.27; 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 и RSA (по умолчанию)
(2) DSA и Elgamal
(3) DSA (только для подписи)
(4) RSA (только для подписи)
(14) Имеющийся на карте ключ
Ваш выбор?
```

Рис. 3.7: Создание ключа PGP

4) Добавляем ключ PGP в GitHub

```
nastarkov@dk8n67 ~ $ gpg --list-secret-keys --keyid-format LONG
```

Рис. 3.8: Добавление ключа

Копируем сгенерированный ключ PGP ключ в буфер обмена

```
nastarkov@dk8n67 ~ $ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/n/a/nastarkov/.ssh/id_rsa):
```

Рис. 3.9: Копирование ключа в буфер обмена

Далее создаем сам ключ в гитхабе



Рис. 3.10: Добавление ключа на сайте

5) Настраиваем автоматические подписи коммитов git

```
nastarkov@dk8n67 ~ $ git config --global user.signkey 1FEEEBE8C018D76
nastarkov@dk8n67 ~ $ git config --global commit.gpgsign true
nastarkov@dk8n67 ~ $ git config --global gpg.program $(which gpg2)
```

Рис. 3.11: Автоматические подписи git

6) Создаем SSH ключ

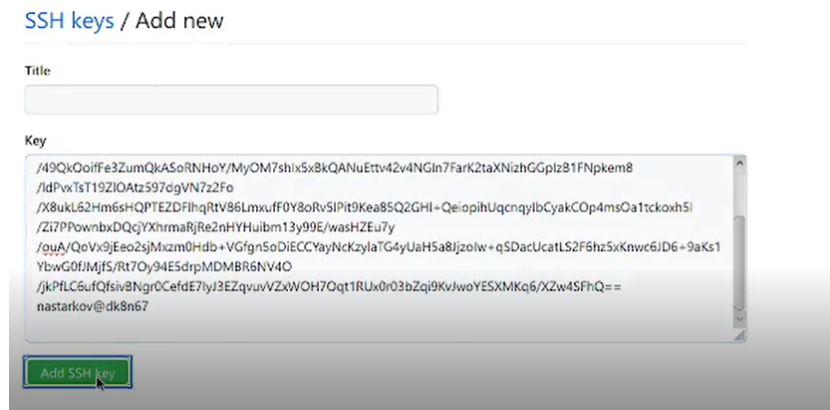


Рис. 3.12: Добавление ssh ключа

7) Создаем каталог с названием, которое требуется и переходим в него

```
nastarkov@dk8n67 ~ $ mkdir -p ~/work/study/2021-2022/"Операционные системы"
nastarkov@dk8n67 ~ $ cd ~/work/study/2021-2022/"Операционные системы"
```

Рис. 3.13: Создание каталога

3.3 Создание рабочей среды

Клонируем шаблон рабочей среды в свой гитхаб

```
nastarkov@dkn67 ~/work/study/2021-2022/Операционные системы $ git clone --recursive git@github.com:yamadharma/course-direct
ory-student-template.git os-intro
Клонирование в «os-intro»...
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 35 (delta 7), reused 34 (delta 0), pack-reused 0
Получение объектов: 100% (35/35), 15.77 КиБ | 5.26 МБ/с, готово.
Определение изменений: 100% (7/7), готово.
Подмодуль «template/presentation» (https://github.com/yamadharma/academic-presentation-markdown-template.git) зарегистрирова
н по пути «template/presentation»
Подмодуль «template/report» (https://github.com/yamadharma/academic-laboratory-report-template.git) зарегистрирован по пути
«template/report»
Клонирование в «/afs/dk.sci.pfu.edu.ru/home/n/a/nastarkov/work/study/2021-2022/Операционные системы/os-intro/template/prese
ntation»...
remote: Enumerating objects: 42, done.
remote: Counting objects: 100% (42/42), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 42 (delta 9), reused 40 (delta 7), pack-reused 0
Получение объектов: 100% (42/42), 31.19 КиБ | 912.00 КиБ/с, готово.
Определение изменений: 100% (9/9), готово.
Клонирование в «/afs/dk.sci.pfu.edu.ru/home/n/a/nastarkov/work/study/2021-2022/Операционные системы/os-intro/template/repor
t»...
```

Рис. 3.14: Клонирование рабочей среды

Настраиваем каталог курса(удаляем лишние файлы, создаем необходимые каталоги, отправляем файлы на сервер)

```
nastarkov@dkn67 ~/work/study/2021-2022/Операционные системы $ cd ~/work/study/2021-2022/"Операционные системы"/os-intro
nastarkov@dkn67 ~/work/study/2021-2022/Операционные системы/os-intro $ rm package.json
nastarkov@dkn67 ~/work/study/2021-2022/Операционные системы/os-intro $ makeCOURSE=os-intro
nastarkov@dkn67 ~/work/study/2021-2022/Операционные системы/os-intro $ git add
bash: 2git: команда не найдена
nastarkov@dkn67 ~/work/study/2021-2022/Операционные системы/os-intro $ git commit -am'feat(main): make course structure'
```

Рис. 3.15: Настройка каталога курса

Контрольные вопросы:

- 1) Version Control System — программное обеспечение для облегчения работы с изменяющейся информацией. 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) 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 неполную версию изменённых файлов, а производить так называемую дельта-компрессию—сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.

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.

- 3) Централизованные системы — это системы, которые используют архитектуру клиент / сервер, где один или несколько клиентских узлов напрямую подключены к центральному серверу. **Пример** - Wikipedia.

В децентрализованных системах каждый узел принимает свое собственное решение. Конечное поведение системы является совокупностью решений отдельных узлов. **Пример** — Bitcoin.

В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером.

- 4) Создадим локальный репозиторий. Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория:

```
git config --global user.name "Имя Фамилия" git config --global user.email "work@mail"
и настроив utf-8 в выводе сообщений git:
```

```
git config --global quotePath false
```

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

```
cd
mkdir tutorial
cd tutorial
git init
```

- 5) Для последующей идентификации пользователя на сервере репозитория необходимо сгенерировать пару ключей (приватный и открытый):

```
ssh-keygen -C "Имя Фамилия work@mail"
```

Ключи сохраняться в каталоге ~/.ssh/.

Скопировав из локальной консоли ключ в буфер обмена

```
cat ~/.ssh/id_rsa.pub | xclip -sel clip
```

вставляем ключ в появившееся на сайте поле.

- 6) У Git две основных задачи: первая — хранить информацию о всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечивать удобства командной работы над кодом.
- 7) Основные команды git: Наиболее часто используемые команды 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_файлов – удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm имена_файлов – 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 (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) – отправка изменений конкретной ветки в центральный репозиторий: git push origin имя_ветки–слияние ветки стекущим деревом:git merge --no-ff имя_ветки–удаление ветки: – удаление локальной уже слиятой 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) Использование git при работе с локальными репозиториями (добавления текстового документа в локальный репозиторий):

```
git add hello.txt
```

```
git commit -am 'Новый файл'
```

- 9) Проблемы, которые решают ветки git:

- нужно постоянно создавать архивы с рабочим кодом
- сложно “переключаться” между архивами
- сложно перетаскивать изменения между архивами
- легко что-то напутать или потерять

- 10) 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 списки меняющихся 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
```

## 4 Выводы

**Вывод:** в ходе выполнения лабораторной работы я изучил идеологию и применение средств контроля версий, освоил умения по работе с git.