

Отчет по лабораторной работе номер 1

Хамбалеев Булат Галимович

Содержание

1	Цель работы	5
2	Задание	6
3	Теория	7
4	Выполнение работы	8
5	Библиография	12
6	Выводы	13

List of Tables

List of Figures

4.1	рис.1. Аккаунтна на GitHub.	8
4.2	рис.2. Имя пользователя и почта.	8
4.3	рис.3. Создание ключа.	9
4.4	рис.4. Получение ключа.	9
4.5	рис.5. Ввод ключа.	9
4.6	рис.6. Создание ключа.	10
4.7	рис.7. Получение ключа.	10
4.8	рис.8. Ввод ключа.	10
4.9	рис.9. Ввод ключа.	11

1 Цель работы

Изучить идеологию и применение средств контроля версий. Освоить умения по работе с git.

2 Задание

Лабораторная работа подразумевает использование git для создания репозитория для лабораторных работ.

3 Теория

Git (произносится «гит») — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано.

4 Выполнение работы

1. Создадим базовую конфигурацию для работы с git.(рис 1-2)

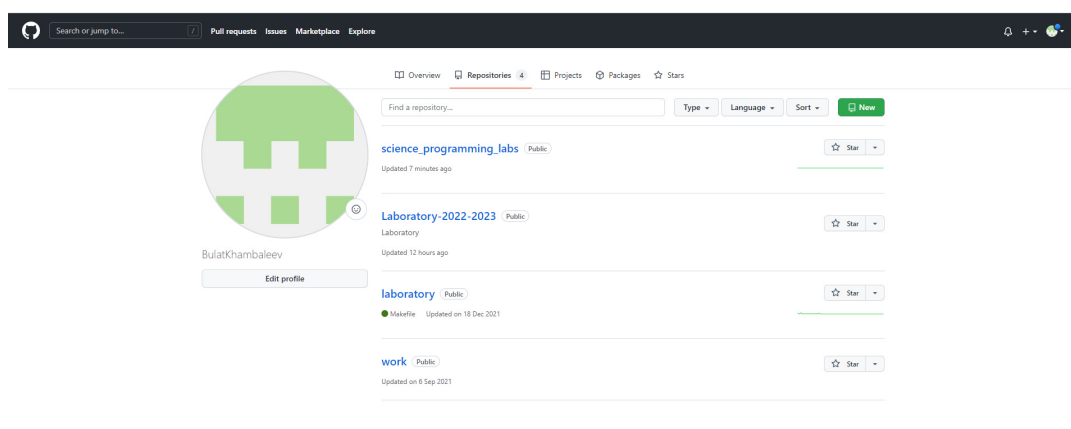


Figure 4.1: рис.1. Аккаунтна на GitHub.

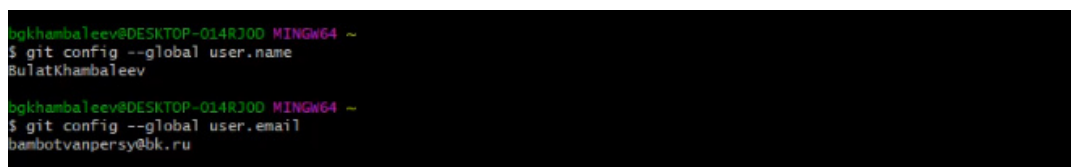




Figure 4.2: рис.2. Имя пользователя и почта.

2. Создадим ключ SSH.(рис 3-5)


Import a repository.


Owner * Repository name *

 BulatKhambaleev / science_programming_labs 

Great repository names are short: science_programming_labs is available. [How about vigilant-enigma?](#)

Description (optional)

☒  Public
Anyone on the internet can see this repository. You choose who can commit.

☐  Private
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ Add a README file
This is where you can write a long description for your project. [Learn more.](#)

Add README

Figure 4.6: рис.6. Создание ключа.

```
lgkhambaleev@DESKTOP-014R300 MINGW64 ~/.ssh
$ cd ~

lgkhambaleev@DESKTOP-014R300 MINGW64 ~
$ git clone git@github.com:BulatKhambaleev/science_programming_labs.git
Cloning into 'science_programming_labs'...
warning: You appear to have cloned an empty repository.

lgkhambaleev@DESKTOP-014R300 MINGW64 ~
$ cd science_programming_labs

lgkhambaleev@DESKTOP-014R300 MINGW64 ~/science_programming_labs (main)
$ ls

lgkhambaleev@DESKTOP-014R300 MINGW64 ~/science_programming_labs (main)
$ ls -a
./ ../ .git/
```

Figure 4.7: рис.7. Получение ключа.

```
lgkhambaleev@DESKTOP-014R300 MINGW64 ~/science_programming_labs (main)
$ git commit -m "first commit"
[main (root-commit) e53bcfd] first commit
1 file changed, 1 insertion(+)
create mode 100644 README.md

lgkhambaleev@DESKTOP-014R300 MINGW64 ~/science_programming_labs (main)
$ git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 292 bytes | 292.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:BulatKhambaleev/science_programming_labs.git
 * [new branch]      main -> main
```

Figure 4.8: рис.8. Ввод ключа.

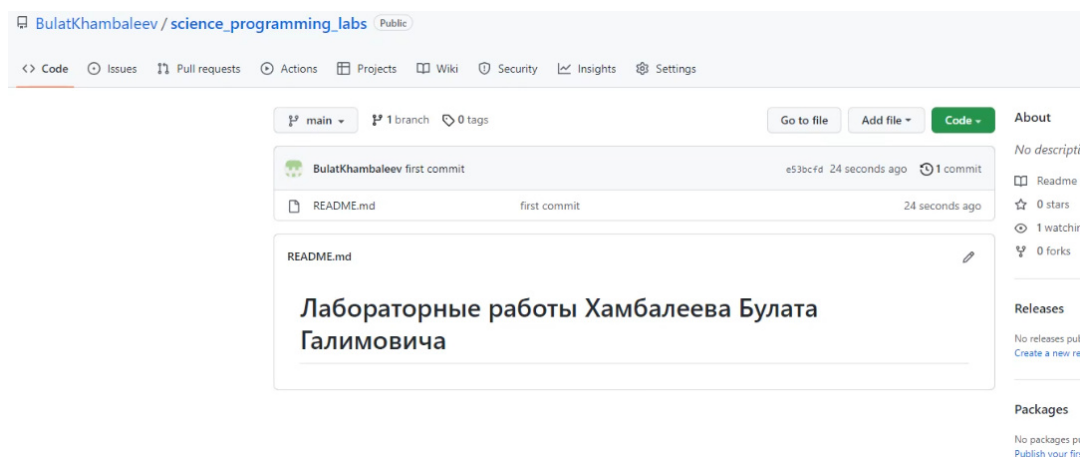


Figure 4.9: рис.9. Ввод ключа.

5 Библиография

1. ТУИС РУДН

6 Выводы

Во время выполнения лабораторной работы я освоил на практике git и создал репозиторий для лабораторных работ.

#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.

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

- Хранилище (repository), или репозиторий, — место хранения всех версий и служебной информации. Commit — синоним версии; процесс создания новой версии. Рабочая копия (working copy) — текущее состояние файлов проекта, основанное на версии, загруженной из хранилища (обычно на последней).

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

- Централизованные системы контроля версий представляют собой приложения типа клиент-сервер, когда репозиторий проекта существует в единственном экземпляре и хранится на сервере. Доступ к нему осуществляется через специальное клиентское приложение. В качестве примеров таких

программных продуктов можно привести CVS, Subversion. распределенные системы контроля версий (Distributed Version Control System, DVCS) позволяют хранить репозиторий (его копию) у каждого разработчика, работающего с данной системой. При этом можно выделить центральный репозиторий (условно), в который будут отправляться изменения из локальных и, с ним же эти локальные репозитории будут синхронизироваться. При работе с такой системой, пользователи периодически синхронизируют свои локальные репозитории с центральным и работают непосредственно со своей локальной копией. После внесения достаточного количества изменений в локальную копию они (изменения) отправляются на сервер. При этом сервер, чаще всего, выбирается условно, т. к. в большинстве DVCS нет такого понятия как “выделенный сервер с центральным репозиторием”. Еще 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) Опишите действия с VCS при единоличной работе с хранилищем.

- 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) Опишите порядок работы с общим хранилищем VCS.

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

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

Ключи сохраняются 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) Каковы основные задачи, решаемые инструментальным средством git?

- У Git две основных задачи: первая — хранить информацию о всех изменениях в вашем коде, начиная с самой первой строки, а вторая — обеспечение удобства командной работы над кодом.

7) Назовите и дайте краткую характеристику командам 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`

– 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`

– отправка изменений конкретной ветки в центральный репозиторий:

`git push origin имя_ветки`

– слияние ветки с текущим деревом:

`git merge --no-ff имя_ветки`

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

• Использование `git` при работе с локальными репозиториями (добавления текстового документа в локальный репозиторий):

`git add hello.txt`

`git commit -am 'Новый файл'`

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

- Ветки нужны для того, чтобы программисты могли вести совместную работу над проектом и не мешать друг другу при этом. Кроме того, с помощью branches решаются следующие проблемы: нужно постоянно создавать архивы с рабочим кодом, сложно “переключаться” между архивами, сложно перетаскивать изменения между архивами, легко что-то напутать или потерять.

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. Во время работы над проектом эти файлы могут создаваться, но их 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
```