Лабораторная работа №3

Отчёт по лабораторной работе №3

Макарова Анастасия Михайловна

Содержание

Цель работы

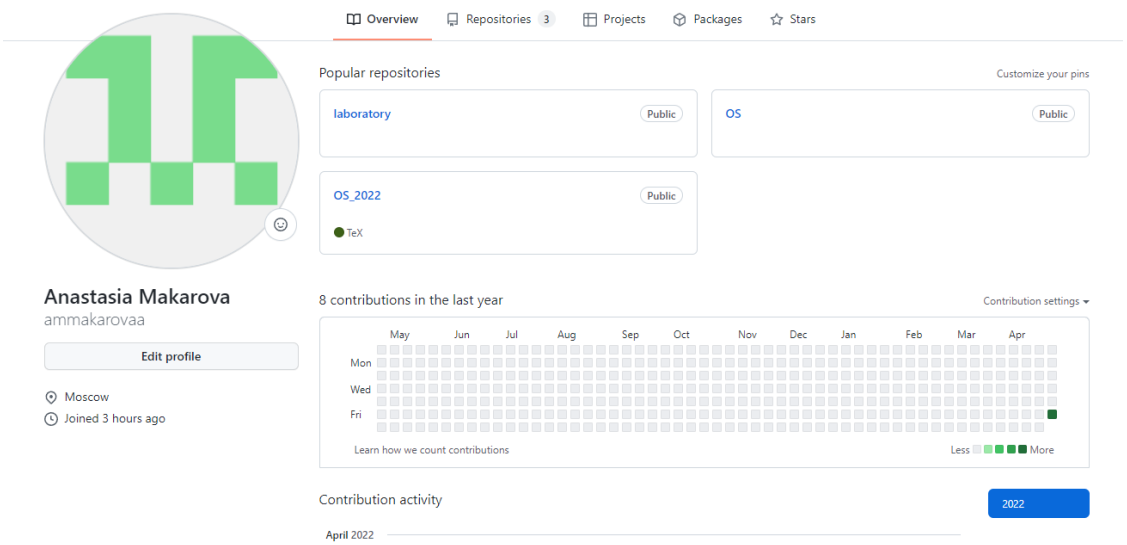
Цель данной лабораторной работы - Научиться оформлять отчёты с помощью легковесного языка разметки Markdown.

Задание

1. Сделайте отчёт по предыдущей лабораторной работе в формате Markdown.
2. В качестве отчёта просьба предоставить отчёты в 3ф орматах: pdf,docx имd (в архиве, поскольку он должен содержать скриншоты, Makefile ит.д.)

Выполнение лабораторной работы

1)Создаем учетную запись на https://github.com. и заполняем основные данные. (Рис.1)



Создание учетной записи

2)Настраиваем систему контроля версий git. Синхронизируем учётную запись github с компьютером: git config –global user.name “Имя Фамилия”, git config –global user.email “work@mail”. (Рис.2)

```
[ammakarova@10 ~]$ git config --global user.name "Anastasia Makarova"
[ammakarova@10 ~]$ git config --global user.email "makarovanasta427@gmail.com"
```

Рис.2

Необходимо задать имя и email владельца репозитория. Настраиваем верификацию и подписание коммитов git. Задаём имя начальной ветки, параметр autocrlf и параметр safecrlf. (Рис.3)

```
[ammakarova@10 ~]$ git config --global core.quotepath false
[ammakarova@10 ~]$ git config --global init.defaultBranch master
[ammakarova@10 ~]$ git config --global core.autocrlf input
[ammakarova@10 ~]$ git config --global core.safecrlf warn
```

Рис.3

3)Сгенерируем ключи ssh и gpg и вставим их в учётную запись github, чтобы привязать компьютер с github. Создаём ключ ssh с помощью команды ssh-keygen -t rsa -b 4096. (Рис.4)

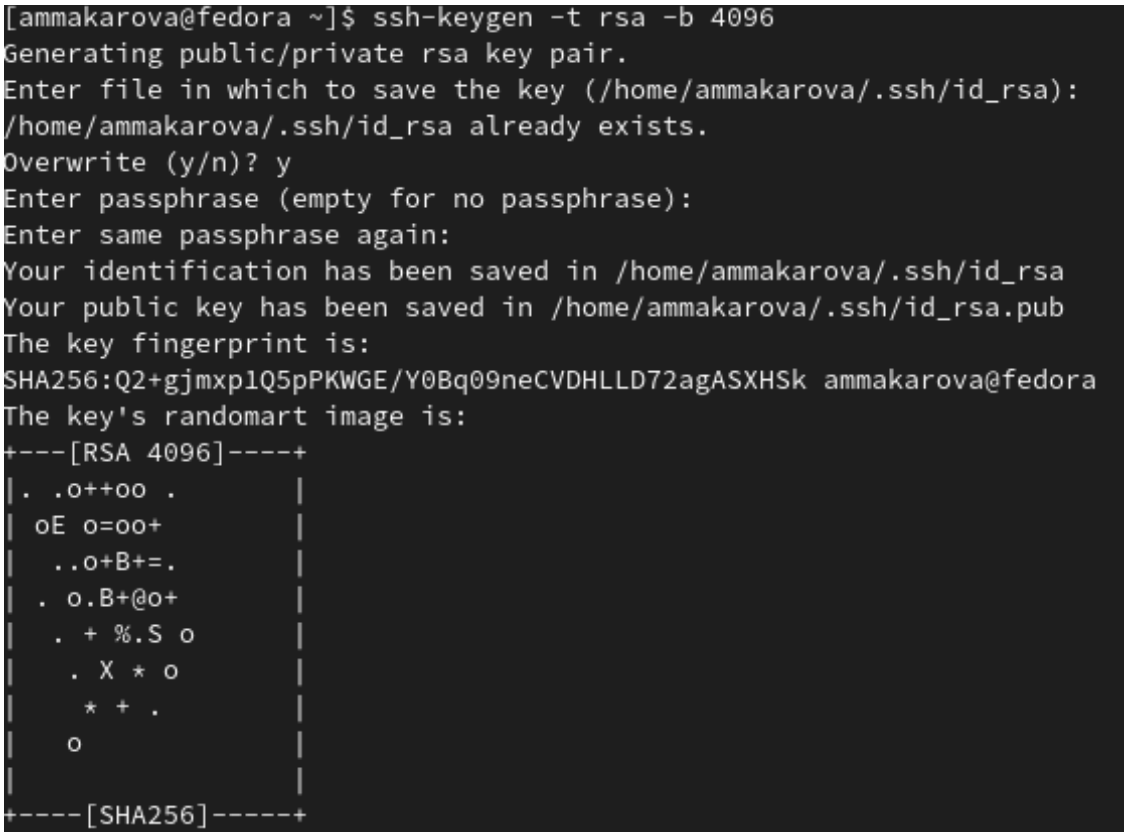


Рис.4

Копируем его в буфер обмена с помощью команды cat ~/.ssh/id_rsa.pub | xclip -sel clip. (Рис.5)

Рис.5

Создаём ключ gpg с помощью команды gpg -full-generate-key и выбираем из предложенных вариантов те, которые указаны в лабораторной работе №2. (Рис.5.1)

```
[ammakarova@10 ~]$ gpg --full-generate-key
gpg (GnuPG) 2.3.4; 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: Anastasia Makarova
Адрес электронной почты: makarovanasta427@gmail.com
Примечание:
Вы выбрали следующий идентификатор пользователя:
    "Anastasia Makarova <makarovanasta427@gmail.com>"
```

Рис.5.1

Выводим список ключей, чтобы скопировать отпечаток приватного ключа. (Рис.6)

```
[ammakarova@fedora ~]$ 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
/home/ammakarova/.gnupg/pubring.kbx
------------------------------------
sec   rsa4096/89E8DBB55CFB29BE 2022-04-30 [SC]
      3B26A5559ECA2DE614BF9A4089E8DBB55CFB29BE
uid                 [  абсолютно ] Anastasia Makarova <makarovanasta427@gmail.com>
ssb   rsa4096/5F740657C0D186C9 2022-04-30 [E]
```

Рис.6

Скопируем сгенерированный gpg ключ в буфер обмена. (Рис.7)

```
[ammakarova@fedora ~]$ gpg --armor --export 89E8DBB55CFB29BE | xclip -sel clip
```

Рис.7

Вставим ключи ssh и gpg в аккаунт github. (Рис.8)

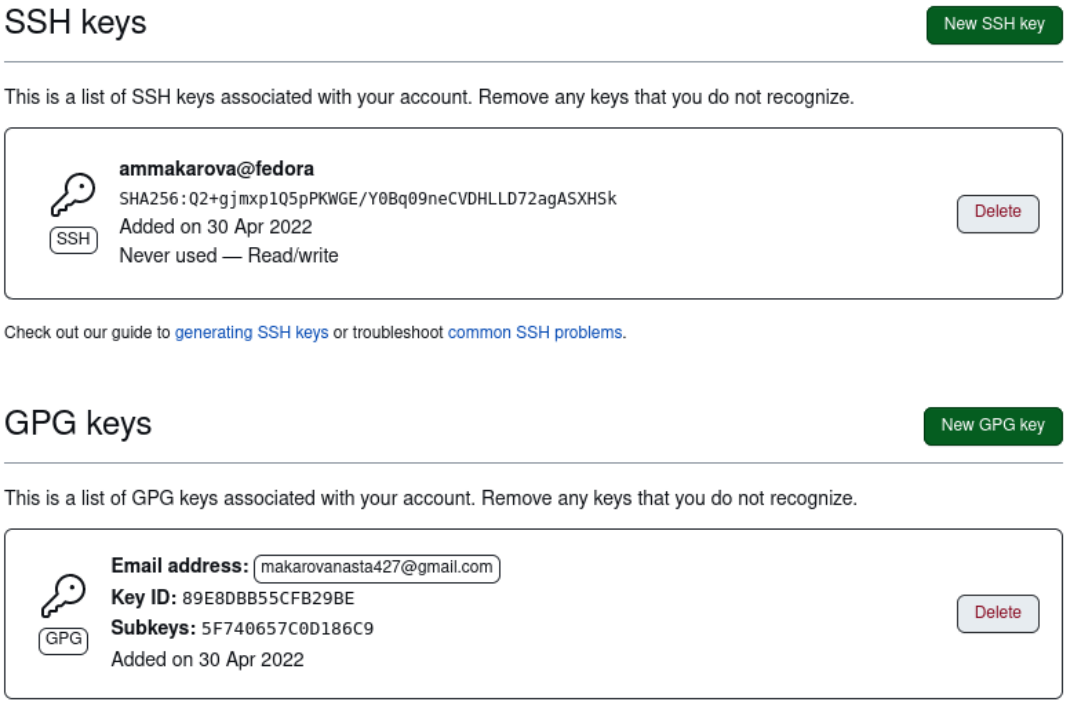


Рис.8

4)Воспользуемся введённым email и указываем git, применяем его при подписи коммитов. (Рис.9)

```
[ammakarova@fedora ~]$ git config --global user.signingkey 89E8DBB55CFB29BE
[ammakarova@fedora ~]$ git config --global commit.gpgsign true
[ammakarova@fedora ~]$ git config --global gpg.program $(which gpg2)
```

Рис.9

5)Создаём путь, в котором будут храниться материалы к лабораторным работам. Перейдём в последнюю папку и скачаем шаблон репозитория. (Рис.10)

```
[ammakarova@10 ~]$ mkdir -p ~/work/study/2021-2022
[ammakarova@10 ~]$ cd ~/work/study/2021-2022
[ammakarova@10 2021-2022]$ mkdir "Операционные системы"
[ammakarova@10 2021-2022]$ cd "Операционные системы"
[ammakarova@10 Операционные системы]$ git clone --recursive https://github.com/yamadharma/course-directory-student-template os-intro
```

Рис.10

6)Создаём репозиторий на github. Создадим необходимые каталоги с помощью команды make COURSE=os-intro и просмотрим содержимое каталога командой ls. (Рис.11)

```
[ammakarova@10 os-intro]$ make COURSE=os-intro
[ammakarova@10 os-intro]$ ls
config  labs  LICENSE  Makefile  package.json  project-personal  README.en.md  README.git-flow.md  README.md  structure  template
```

Рис.11

Перенесём в созданный нами репозиторий все файлы из папки os-intro, затем отправляем все файлы на сервер, чтобы они появились в репозитории github. (Рис.12, 13, 14)

```
[ammakarova@10 os-intro]$ git clone --recursive https://github.com/ammakarovaa/OS_2022.git
Клонирование в «OS_2022»…
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Получение объектов: 100% (3/3), готово.
```

Рис.12

```
[ammakarova@10 OS_2022]$ git add .
[ammakarova@10 OS_2022]$ git commit -am "first"
[main 48f9973] first
 183 files changed, 20098 insertions(+), 1 deletion(-)
 create mode 100644 LICENSE
 create mode 100644 Makefile
 create mode 100644 README.en.md
 create mode 100644 README.git-flow.md
 create mode 100644 config/course/os-intro
 create mode 100644 config/course/sciprog-intro
 create mode 100755 config/script/lab
 create mode 100755 config/script/project-group
 create mode 100755 config/script/project-personal
 create mode 100644 labs/lab01/presentation/Makefile
 create mode 100644 labs/lab01/presentation/presentation.md
 create mode 100644 labs/lab01/report/Makefile
 create mode 100644 labs/lab01/report/bib/cite.bib
 create mode 100644 labs/lab01/report/image/placeimg_800_600_tech.jpg
 create mode 100644 labs/lab01/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
 create mode 100644 labs/lab01/report/report.md
 create mode 100644 labs/lab02/presentation/Makefile
 create mode 100644 labs/lab02/presentation/presentation.md
 create mode 100644 labs/lab02/report/Makefile
 create mode 100644 labs/lab02/report/bib/cite.bib
 create mode 100644 labs/lab02/report/image/placeimg_800_600_tech.jpg
 create mode 100644 labs/lab02/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
 create mode 100644 labs/lab02/report/report.md
 create mode 100644 labs/lab03/presentation/Makefile
 create mode 100644 labs/lab03/presentation/presentation.md
 create mode 100644 labs/lab03/report/Makefile
 create mode 100644 labs/lab03/report/bib/cite.bib
 create mode 100644 labs/lab03/report/image/placeimg_800_600_tech.jpg
```

Рис.13

```
[ammakarova@10 OS_2022]$ git push
Username for 'https://github.com': Anastasia Makarova
Password for 'https://Anastasia Makarova@github.com':
Перечисление объектов: 50, готово.
Подсчет объектов: 100% (50/50), готово.
При сжатии изменений используется до 2 потоков
Сжатие объектов: 100% (44/44), готово.
Запись объектов: 100% (48/48), 281.07 КиБ | 8.27 МиБ/с, готово.
Всего 48 (изменений 3), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/ammakarovaa/OS_2022.git
   511e4c8..48f9973  main -> main
```

Рис.14

7)Откроем наш репозиторий на github и убедимся в праивльности выполненных действий. (Рис.15)

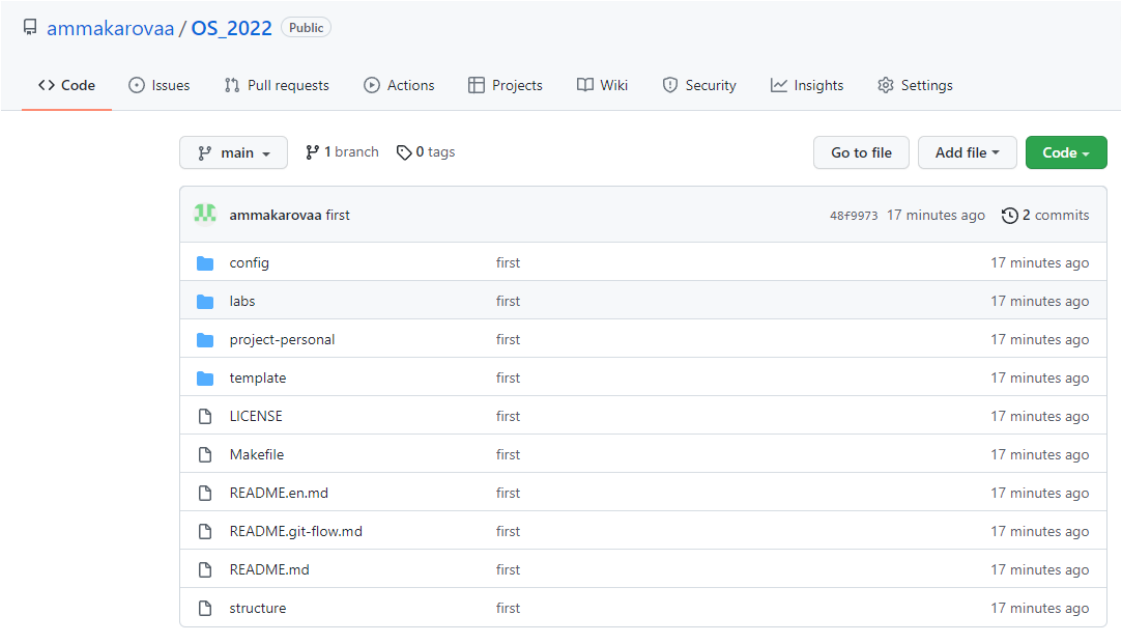


Рис.15

Выводы

В данной лабораторной работе я научилась оформлять отчёты с помощью легковесного языка разметки Markdown.

Ответы на контрольные вопросы.

1. Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды git с различными опциями. 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 определённыхкоманд получает нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляютсяиз 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,чтопозволяет 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. Крометого, 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.
4. 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 дельтакомпрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.

5. 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.
6. У Git две основных задачи: первая — хранить информацию о всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение 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 изменения: 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 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. Использования 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.