Лабораторная работа 2

Скрипникова София

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

В прошлом семестре мною была выполнена привязка git, все функционирует исправно (рис. 2.1).

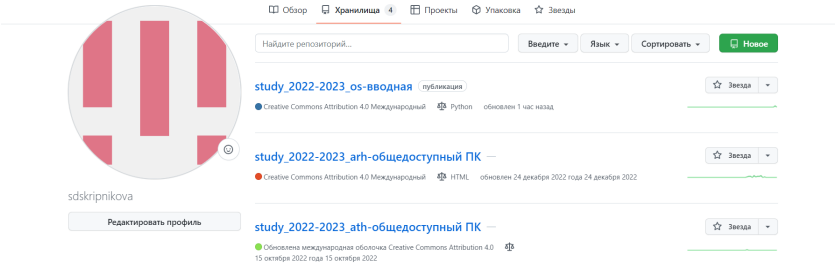


Рис. 2.1: Исправное функционирование

Генерирование ключей PGP и их настройка (рис. **??**).

```
[sdskripnikova@fedora ~]$ 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: Sofiya
Адрес электронной почты: nice.sofy@mail.ru
Примечание:
Вы выбрали следующий идентификатор пользователя:
    "Sofiya <nice.sofy@mail.ru>"
```

{#fig:002 width=70%}

Далее выводим список ключей (рис. **??**).

```
[sdskripnikova@fedora ~]$ gpg --list-secret-key --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/sdskripnikova/.gnupg/pubring.kbx
------------------------------------
sec   rsa4096/EE6EBFE6284FE35D 2023-02-17 [SC]
      BB2A79854469731AACD906FBEE6EBFE6284FE35D
uid                 [  абсолютно ] Sofiya <nice.sofy@mail.ru>
ssb   rsa4096/342AD478F20FD8E0 2023-02-17 [E]

[sdskripnikova@fedora ~]$
```

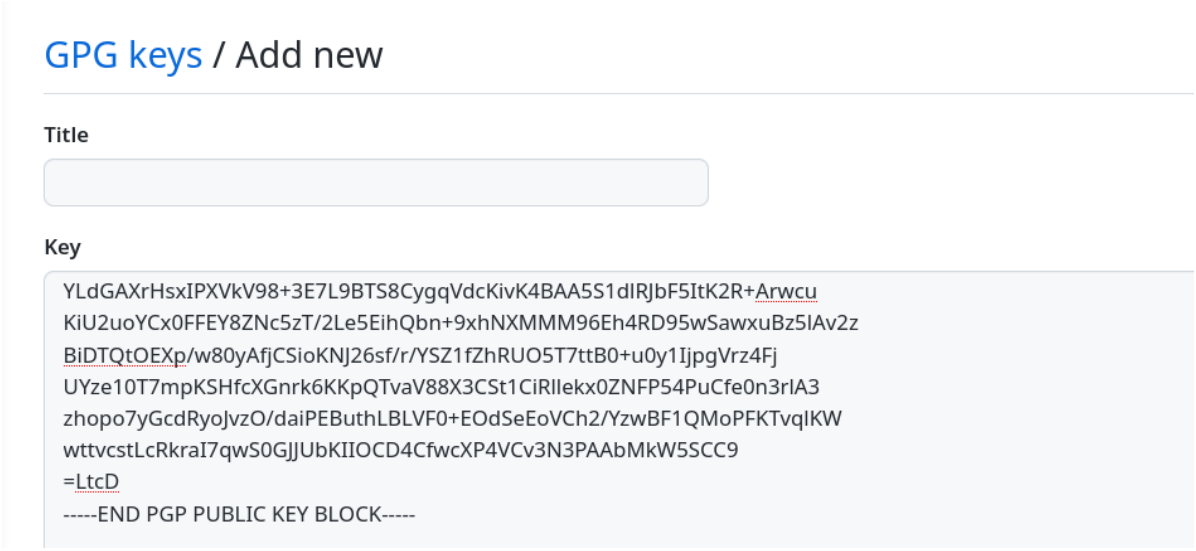
{#fig:003 width=70%}

Копируем наш сгенерированный PGP ключ в буфер обмена (рис. **??**).

```
[sdskripnikova@fedora ~]$ gpg --armor --export EE6EBFE6284FE35D | xclip -sel clip
[sdskripnikova@fedora ~]$
```

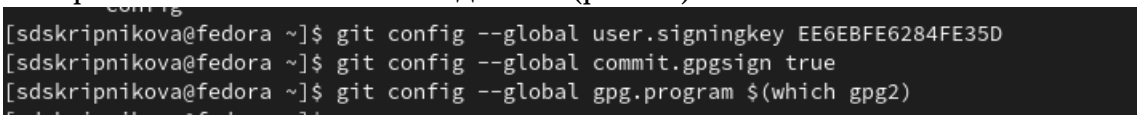
{#fig:004 width=70%}

Переходим в настройки GitHub и вставляем полученный ключ в поле ввода (рис. **??**).



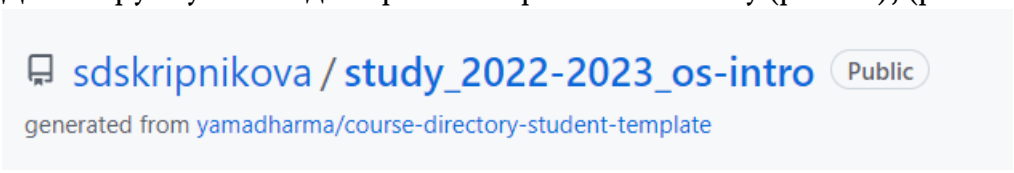
{#fig:005 width=70%}

Настройка автоматических подписей (рис. **??**).



{#fig:005 width=70%}

Далее вручную я создала репозиторий по шаблону (рис. **??**); (рис. **??**).



{#fig:007 width=70%}

{#fig:008 width=70%}

#Контрольные вопросы

1) Система контроля версий (Version Control System, VCS) – программное обеспечение для облегчения работы с изменяющейся информацией. VCS поз-

воляет 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. 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. 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.

2) Хранилище (репозиторий) – в нем хранятся все документы вместе с историей их изменения и другой служебной информацией. Рабочую копию необходимо периодически синхронизировать с репозиторием, эта операция предполагает отправку в него изменений, которые пользователь внес в свою рабочую копию. За это отвечает команда commit. С помощью этой команды можно также передать комментарий о сделанных изменениях. Ис-

тория – список предыдущих изменений/коммитов. Рабочая копия – копия проекта, связанная с репозиторием, в которой непосредственно работает пользователь.

3) Централизованные VCS Клиент-серверная модель: один центральный репозиторий, с которым разработчики взаимодействуют по сети. Примеры: ⊠ CVS ⊠ Subversion (SVN) ⊠ Perforce Децентрализованная VCS Клиенты полностью копируют репозиторий, вместо простого скачивания снимка всех файлов (состояния файлов в определенный момент времени). В этом случае, если сервер выйдет из строя, то клиентский репозиторий можно будет скопировать на другой, рабочий, сервер, ведь данный репозиторий является полным бэкапом всех данных.Примеры: ⊠ Git ⊠ Mercurial ⊠ Bazaar
4) Действия с VCS при единоличной работе с хранилищем: ⊠ Создать новый репозиторий на локальном устройстве (если он не был создан), ⊠ Внести изменения в исходные файлы, ⊠ Выполнить индексацию необходимых файлов, ⊠ Проверить внесенные изменения, ⊠ Выполнить commit, ⊠ Отправить локальный репозиторий на удаленный сервер, при необходимости.
5) Действия при работе с общим хранилищем VCS: ⊠ Обычно проект уже создан и его нужно загрузить из общего удаленного хранилища, ⊠ Необходимо создать свою рабочую ветку, ⊠ Внести изменения внутри своей рабочей ветки, ⊠ Выполнить индексацию необходимых файлов на своем локальном устройстве, ⊠ Проверить внесенные изменения, ⊠ Выполнить commit, ⊠ Свою рабочую ветку отправить в общее хранилище, ⊠ При необходимости внести изменения и отправить снова, ⊠ После администратор объединит вашу ветку с master branch.
6) Git – это система управления версиями. 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. Git запоминает не все изменения, а только те, которые вы скажите. Обычно работа с Git выглядит так: ⊠ сверстали шапку сайта – сделали commit; ⊠

сверстали контент страницы – сделали второй commit; ⊠ закончили верстать страницу – сделали третий commit и отправили код на сервер, чтобы вашу работу могли увидеть коллеги, либо чтобы опубликовать страницу с помощью Capistrano.

7) 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: o добавить все изменённые и/или созданные файлы и/или каталоги: git add . o добавить конкретные изменённые и/или созданные файлы и/или каталоги: git add имена_файлов o 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: o сохранить все добавленные изменения и все изменённые файлы: git commit -am ‘Описание коммита’ o 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 с текущим деревом: git merge –no-ff имя_ветки ⊠ удаление ветки: o удаление локальной уже слитой с основным деревом ветки: git branch -d имя_ветки o принудительное удаление локальной ветки: git branch -D имя_ветки o удаление ветки с центрального репозитория: git push origin :имя_ветки
8) Примеры использования VCS при работе с локальными репозиториями: Создание небольшого проекта на своем локальном устройстве, работа с

файлами (например, каталог, содержащий документы, презентации, которые будут часто редактироваться), работа с фотографиями, видео и т.д. Примеры использования VCS при работе с удаленными репозиториями: Примеры могут быть те же, но теперь над ними работают несколько человек. Такая система позволяет следить за работой других пользователей.

9) Ветвление («ветка», branch) – один из параллельных участков истории в одном хранилище, исходящих из одной версии (точки ветвления). ☒ Обычно есть главная ветка (master), или ствол (trunk). ☒ Между ветками, то есть их концами, возможно слияние. Ветки используются для разработки одной части функционала изолированно от других. Каждая ветка представляет собой отдельную копию кода проекта. Ветки позволяют одновременно работать над разными версиями проекта. Каждый репозиторий по-умолчанию имеет ветку master. Всякий раз, когда требуется разработка нового функционала, не внося при этом изменений в основную рабочую версию, можно создавать новую ветку, на основе рабочей, и вести разработку в ней – новой копии кода проекта. Когда функционал доделан и оттестирован, можно сделать merge – слить отдельную ветку обратно с основной. При слиянии этой временной ветки в основную, все её коммиты разом перенесутся из одной в другую. Ветвление позволяет обеспечить процесс, при котором всегда в наличии будет рабочая версия проекта, в которой не будет частично завершённого функционала находящегося в активной разработке или же непротестированных фич.
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 список имеющихся шаблонов: curl -L -s https://www.gitignore.io/api/list Затем скачать шаблон, например,

для C и C++

3 Выводы

Я освоила изучила идеологию и применение средств контроля версий

Список литературы