Отчёт по лабораторной работе №3

Операционные системы

Сячинова Ксения Ивановна

Содержание

# 1 Цель работы

Цель работы:научиться оформлять отчёты с помощью легковесного языка разметки Markdown, а также познакомиться с основными возможностями разметки Markdown.

# 2 Задание

Сделайте отчёт по предыдущей лабораторной работе в формате Markdown. В качестве отчёта просьба предоставить отчёты в 3 форматах: pdf, docx и md (в архиве, поскольку он должен содержать скриншоты, Makefile и т.д.)

# 3 Выполнение лабораторной работы

1. Создаём учётную запись на github.(рис. 1)

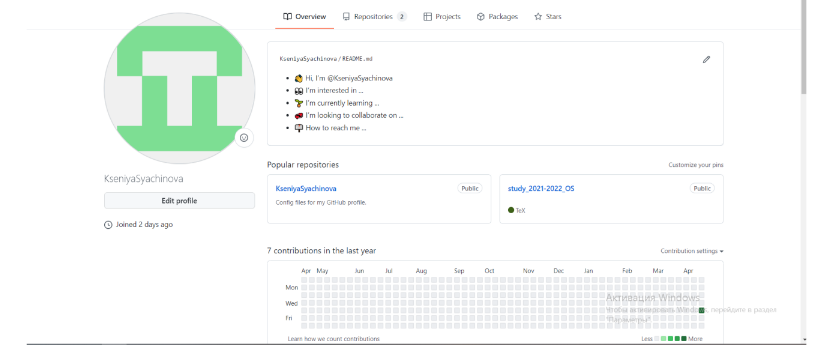


Рис. 1: Создание учётной записи

1. Сделаем предварительную конфигурацию, указав имя и email владельца репозитория с помощью git config —global user.name”Имя Фамилия”, git config —global user.email”work@mail“. (рис. 2)

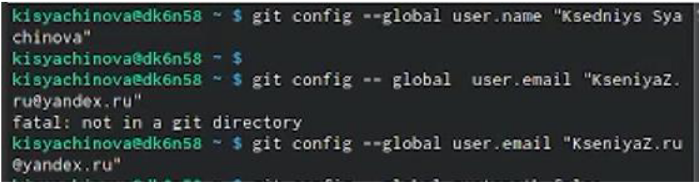


Рис. 2: Делаем конфигурацию

1. После этого создаём новый ключ на github (команда ssh-keygen - C”KseniyaSyachinova [KseniyaZ.ru@yandex.ru](mailto:KseniyaZ.ru@yandex.ru) и привязываем его к копьютеру через консоль. После этого, скопировав из локальной консоли ключ в буфер обмена, вставляем ключ в появившееся на сайте поле. (рис. 3)

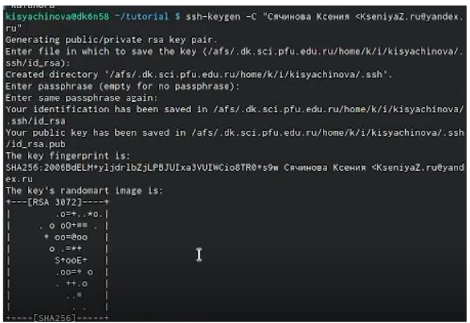


Рис. 3: Делаем конфигурацию

1. Приступаем к базовой настройке git.Зададим имя и email владельца репозитория: git config –globaluser.name “Name Surname” ,git config –global user.email “work@mail”. Настроим utf-8 в выводе сообщений git: git config –global core.quotepath false.Настроим верификацию и подписание коммитов git. Зададим имя начальной ветки: git config –global init.defaultBranch master.Параметр autocrlf: git config –global core.autocrlf input. Параметр safecrlf: git config –global core.safecrlf warn.(рис. 4)

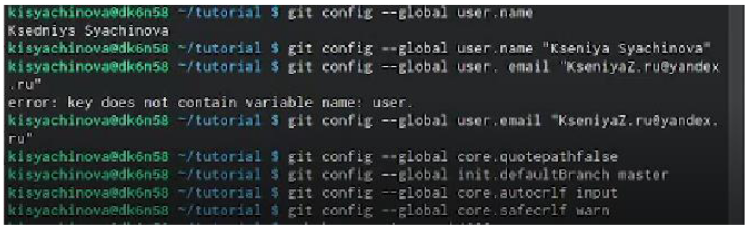


Рис. 4: Настройка git

1. Затем создаём свой ключ по алгоритму: ssh-keygen -t rsa-b 4096 иssh-keygen -t ed25519.(рис. 5)

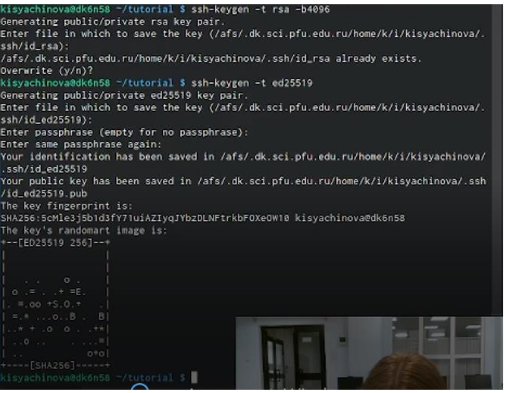


Рис. 5: Создание ssh ключа

1. Создаёмключgpg: gpg –full-generate-key. Затем настраиваем: Тип RSA and RSA Размер 4096 Срок действия,значение по умолчанию — 0 (срок действия не истекает никогда). Имя Адрес электронной почты (рис. 6)

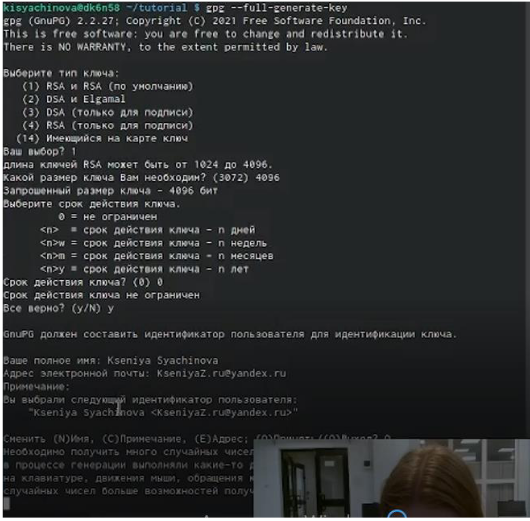


Рис. 6: Создание gpg ключа

1. Добавлем PGP ключ в GitHub. Используем gpg –list-secret-keys –keyid-format LONG. По образцу видим отпечаток моего ключа, вставляем его в следующую конструкцию: gpg--armor--export<pgpFingerprint> | xclip-selclip.Затем перешли в настройки github и вставили полученный ключ. (рис. 7)

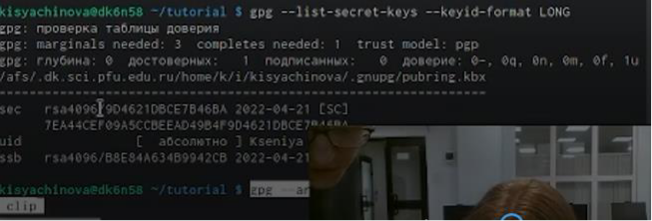


Рис. 7: Добавление ключей

1. Таким образом, у нас получились следующие ssh и gpg ключи: (рис. 8)

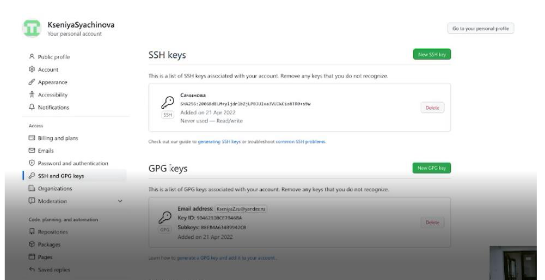


Рис. 8: Полученные ключи

9)Затем настраиваем автоматические подписи коммитов git: git config –global user.signingkey, git config –global commit.gpgsign truel, git config –global gpg.program $(which gpg2) (рис. 9)

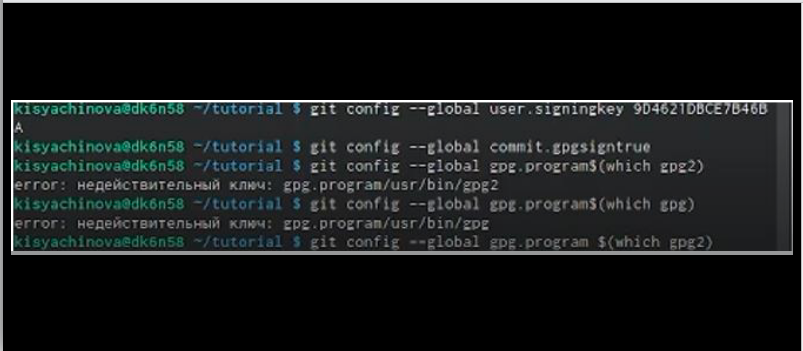


Рис. 9: Настройка коммитов

10)После этого создаём репозиторий курса на основе шаблона. (рис. 10)

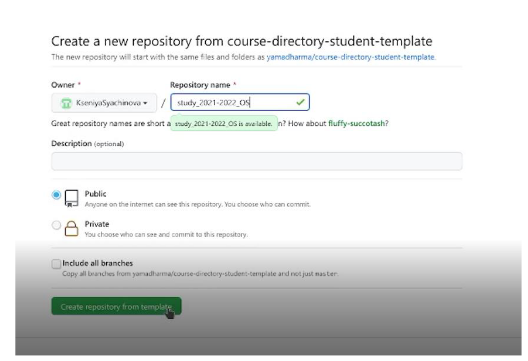


Рис. 10: Содание репозитория

1. Копируем ссылку и с помощью git clone –recursiveдобавляем наши лабораторные работы на github. Впоследствии удаляем лишние файлы: rm package.json, создаём необходимые каталоги: make COURSE=os-intro, и отправляем файлы на сервер: git add ., git commit -am ‘feat(main): make course structure’ , git push. (рис. 11)

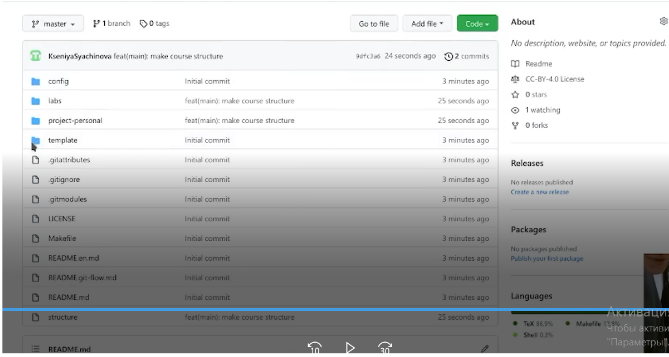


Рис. 11: Итог

# 4 Ответы на контрольные вопросы.

1). Система контроля версий Git представляетсобой набор программ командной строки.Доступк ним можно получить изтерминала посредством ввода командыgitс различ-ными опциями. Системы контроля версий (Version Control System,VCS)применяются при работе нескольких человек над одним проектом. 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 снескольки-ми версиями одного файла,сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Крометого, обычно доступна информация о том, ктоиз 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. Вклассических 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”ИмяФамилия” git config –global user.email”work@mail“и настроив utf-8 в выводе сообщенийgit:git config –global quotepath falseДля инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial, необходимо ввести в командной строке:cdmkdir tutorialcd tutorialgit init 5). Для последующей идентификации пользователя на сервере репозиториевнеобходимо сгенерировать пару ключей (приватный и открытый): ssh-keygen -C”Имя Фамилия [work@mail](mailto:work@mail)“Ключи сохраняться в каталоге~/.ssh/.Скопировав из локальной консоли ключ в буфер обменаcat ~/.ssh/id\_rsa.pub | xclip -sel clipвставляем 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 —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 репози-торий:git push–просмотр списка изменённых файлов втекущей директории:git status–просмотртекущих изменения:git diff–сохранениетекущих 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 (приэтомфайл и/илик аталог 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 натекущей: 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:gitbranch -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 работес локальными репозиториями (добавления текстового документа в локальный репозиторий):git add hello.txtgit commit -am’Новыйфайл 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 проектомтак или иначе могутсоздаваться файлы,которые нетребуется добавлять в последствии в репозиторий. Например, временные файлы, со-здаваемые редакторами,или объектные файлы, создаваемые компиляторами.Можно прописать шаблоны игнорируемых при добавлении в репозиторийтипов файлов в файл.gitignore с помощьюс ервисов. Для этого сначала нужно получить списоки меющихся шаблонов: curl -L -s https://www.gitignore.io/api/listЗатем скачать шаблон,например, для C и C++curl -L -s https://www.gitignore.io/api/c >> .gitignorecurl -L -s https://www.gitignore.io/api/c++ >> .gitigore # Выводы

Я научилась офомлять отчёты в Markdown, познакомилась с основными его возможностями.