Шаблон отчёта по лабораторной работе номер 2

Дисциплина: Операционные системы

Крестененко Полина Александровна

Содержание

# Цель работы

Изучить идеологию и применение средств контроля версий.

# Задание

Написать отчет по 2 лабораторной работе в формате Makefile. Сохранить отчет в трех разных форматах.

# Выполнение лабораторной работы

1. Настройка git • Создаю учетную запись на https://github.com. (рис. -fig. 1)

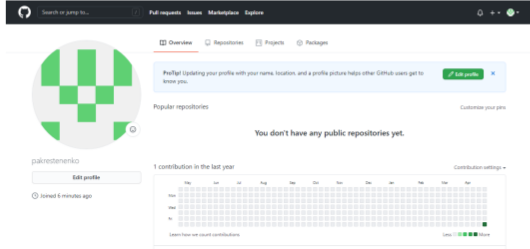


Figure 1: создание учетной записи

• Настраиваю систему контроля версий git. Синхранизирую учётную запись github с компьютером: git config –global user.name “Имя Фамилия” git config –global user.email “work@mail” (рис. -fig. 2)

Figure 2: настройка системы контроля

Figure 2: настройка системы контроля

• Создаю новый ключ на github (команда ssh-keygen -C “pakrestenenko [1032201711@pfur.ru](mailto:1032201711@pfur.ru)”) и привязываю его к копьютеру через консоль.(рис. -fig. 3)

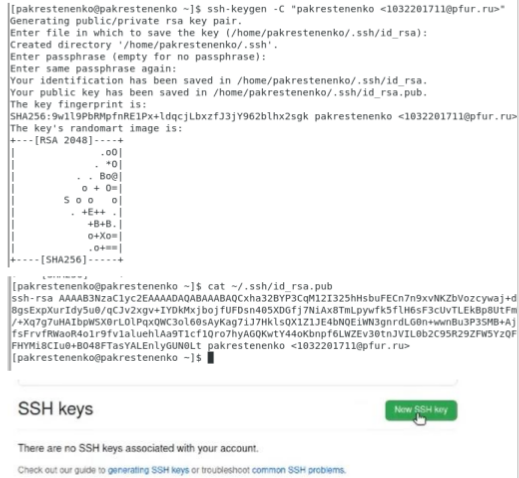


Figure 3: создание нового ключа

(рис. -fig. 4)

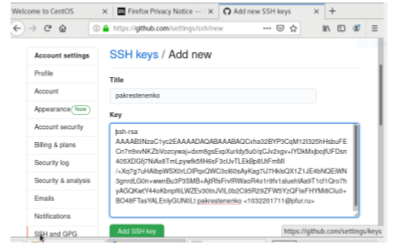


Figure 4: создание нового ключа и привязка

(рис. -fig. 5)



Figure 5: создание нового ключа и привязка

1. Подключение репозитория к github • В githup захожу в «repositories» и создаю новый репозиторий (имя «laborat2», заголовок для файла README). Копируем в консоль ссылку на репозиторий.(рис. -fig. 6)

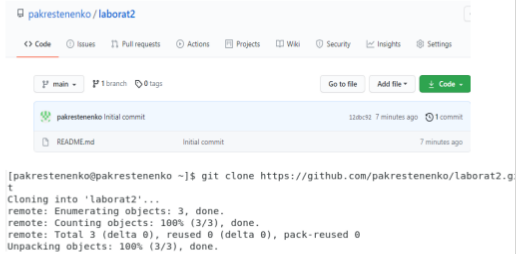


Figure 6: подключение репозитория

• Работаю с каталогом и папками через консоль. Перед тем, как создавать файлы, захожу в репозиорий:(рис. -fig. 7)

Figure 7: работаю с каталогом

Figure 7: работаю с каталогом

Создаю файлы:(рис. -fig. 8)

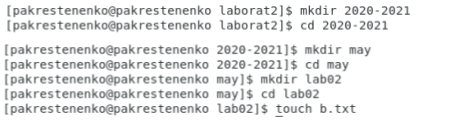


Figure 8: создание файлов

• Добавляю первый коммит и выкладываю на githup. Для того, чтобыправильно разместить первый коммит, необходимо добавить команду git add . , далее с помощью команды git commit -m “first commit” выкладываем коммит:(рис. -fig. 9)

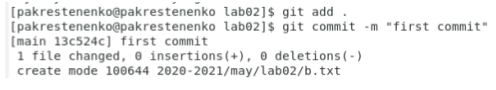


Figure 9: добавляем комминт

• Сохраняю первый коммит (git push):(рис. -fig. 10)

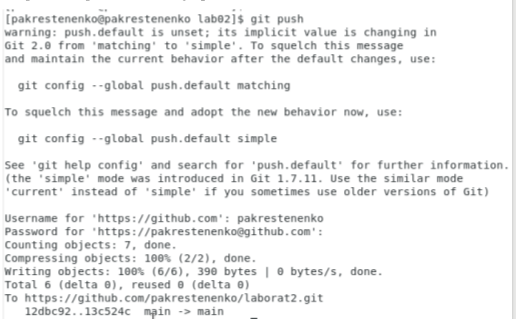


Figure 10: сохранение первого коммита

1. Первичная конфигурация • Добавляю файл лицензии:(рис. -fig. 11)

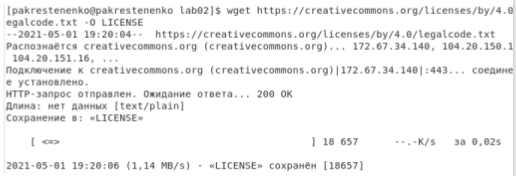


Figure 11: добавление файла лицензии

• Добавляю шаблон игнорируемых файлов. Получаю список имеющихся шаблонов (на скрине представлены не все шаблоны)(рис. -fig. 12)



Figure 12: добавление шаблона игнорируемых файлов

• Скачиваю шаблон, например, для C. Также добавляю новые файлы и выполняю коммит:(рис. -fig. 13)



Figure 13: скачивание шаблона для С

• Отправляю на github (git push):(рис. -fig. 14)

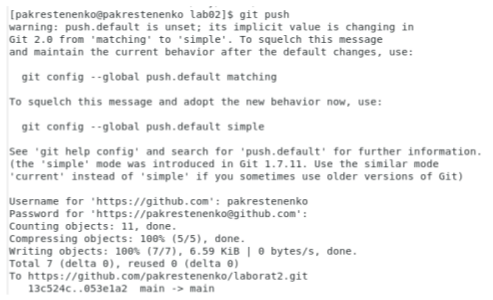


Figure 14: отправляю на гитхаб

1. Конфигурация git-flow • Инициализирую git-flow, используя команду git flow init -f (префикс для ярлыков установлен в v):(рис. -fig. 15)

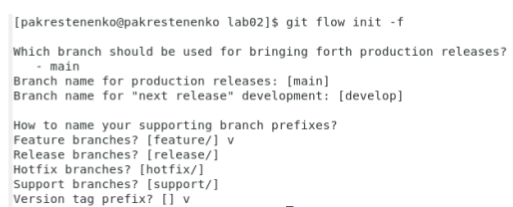


Figure 15: конфигурация

• Проверяю, что нахожусь на ветке develop (git branch):(рис. -fig. 16)

Figure 16: проверка

Figure 16: проверка

• Создаю релиз с версией 1.0.0:(рис. -fig. 17)



Figure 17: создание релиза

• 4. Записываю версию и добавляю в индекс: echo“1.0.0” >> VERSION git add . git commit -am ‘chore(main): add version’(рис. -fig. 18)



Figure 18: записываю видео

• Заливаю релизную ветку в основную ветку (команда git flow release finish1.0.0):(рис. -fig. 19)

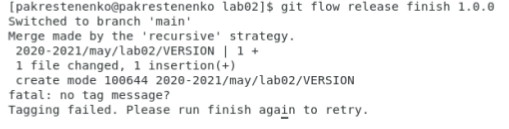


Figure 19: заливаю релизную ветку

• Отправляю данные на github: git push - -all git push - -tags(рис. -fig. 20)

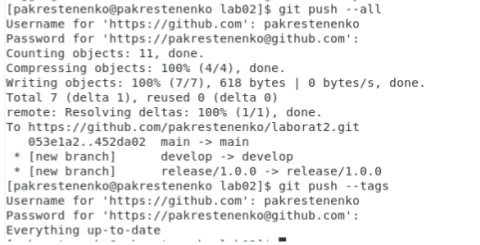


Figure 20: отправка данных на гитхаб

• Создаю релиз на github. Заходим в «Releases», нажимаю «Создать новый релиз». Захожу в теги и заполняю все поля (теги для версии 1.0.0). После создания тега, автоматически сформируется релиз.(рис. -fig. 21)

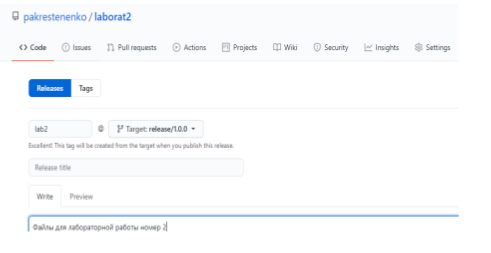


Figure 21: заливаю релизную ветку

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. 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 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 репозиториев необходимо 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 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 — обеспечение удобства командной работы над кодом. 7. Основные команды git: o создание основного дерева репозитория: git init o получение обновлений (изменений)текущего дерева из центрального репозитория: git pull o отправка всех произведённых изменений локального дерева в центральный репозиторий: git push o просмотр списка изменённых файлов в текущей директории: git status o просмотр текущих изменения: git diff 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 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 o создание новой ветки, базирующейся натекущей: git checkout -b имя\_ветки o 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) o отправка изменений конкретной ветки в центральный репозиторий: git push origin имя\_ветки o слияние ветки стекущим деревом: git merge –no-ff имя\_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. Использования git при работе с локальными репозиториями (добавления текстового документа в локальный репозиторий): git add hello.txt git commit -am ‘Новый файл’ 9. 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 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

# Выводы

В ходе выполнения лабораторной работы я изучила идеологию и применение средств контроля версий