Отчёт по лабораторной работе №3

Дисциплина: Операционные системы

Дупленских Василий Викторович

Содержание

Список иллюстраций

Список таблиц

Цель работы

Изучени идеологии и применения средств контроля версий и освоений умений работать с git

Выполнение лабораторной работы.

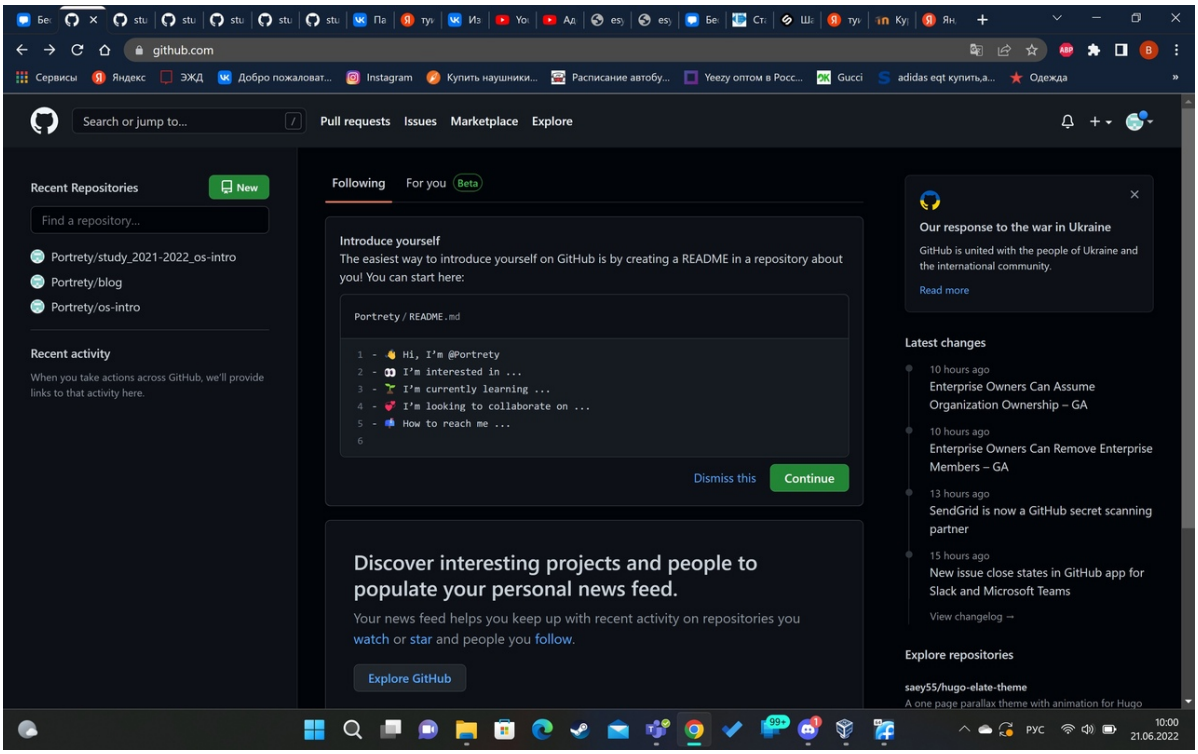


Рис. 0.1: страничка git

1. Зададим имя и email владельца репозитория, кодировку и прочие параметры, Создаю аккаунт на github.

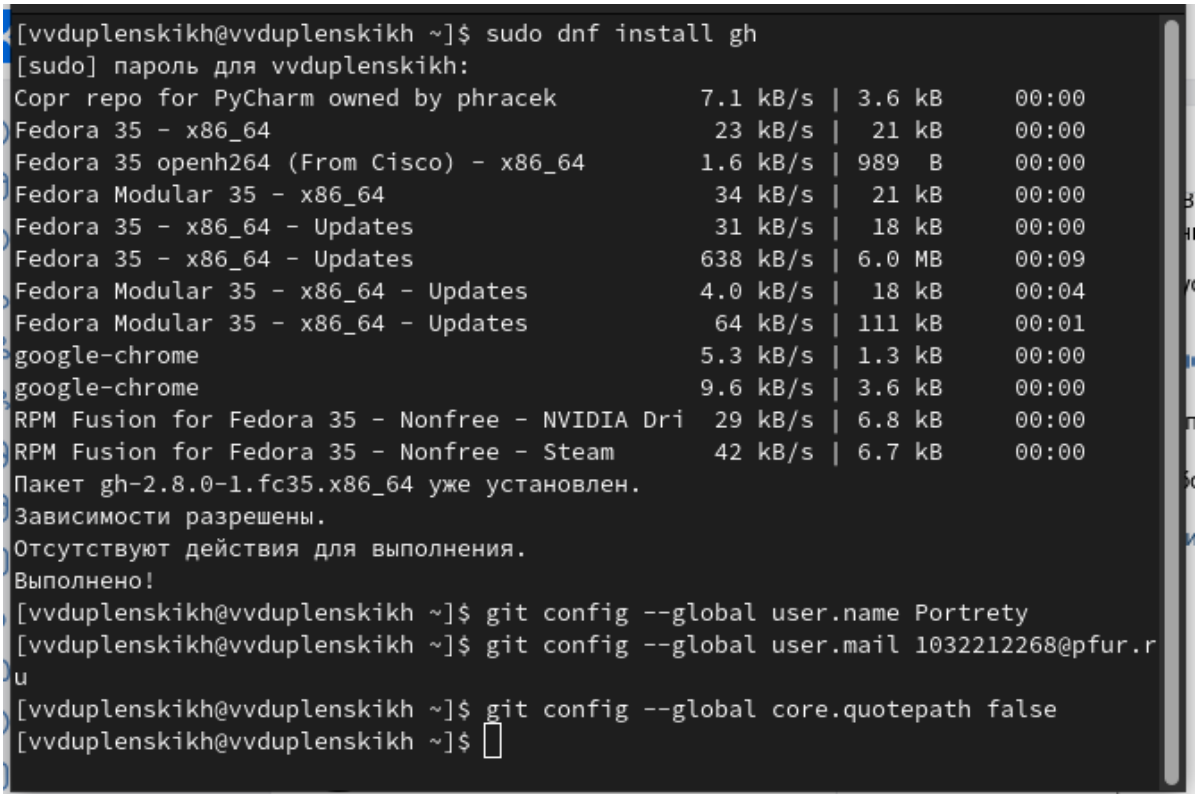
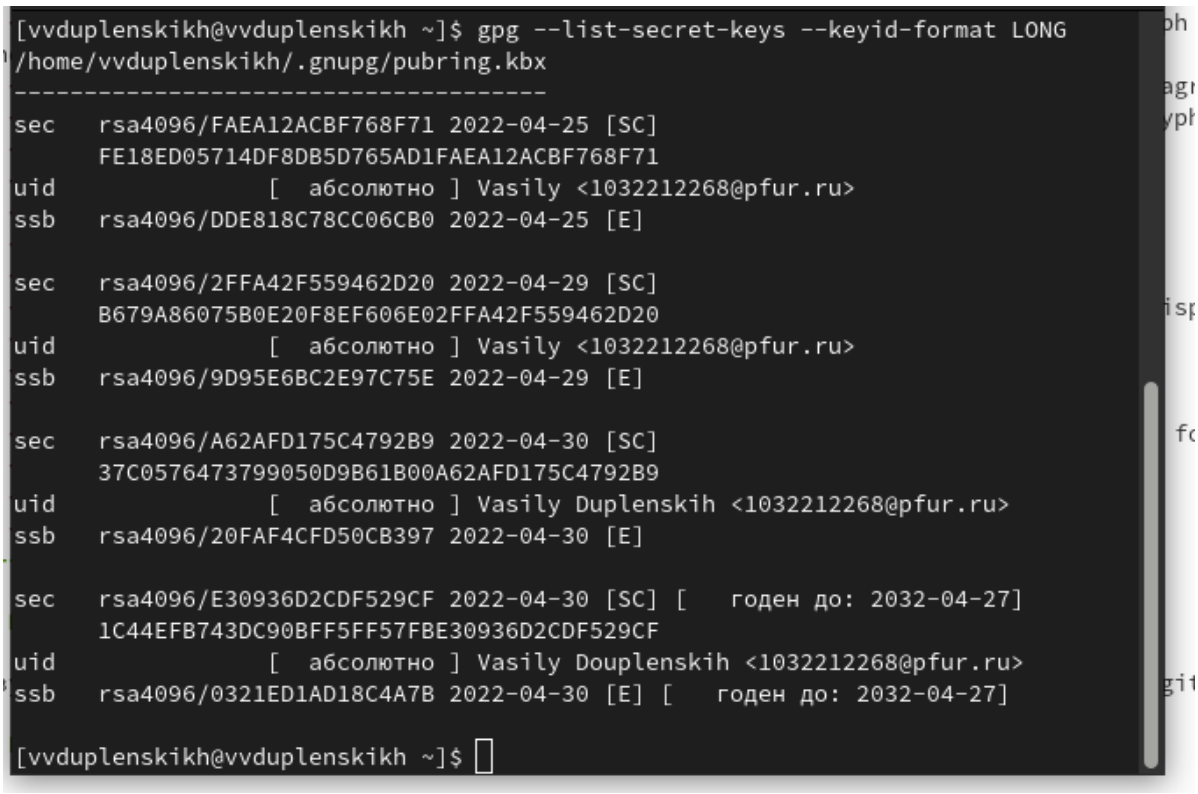
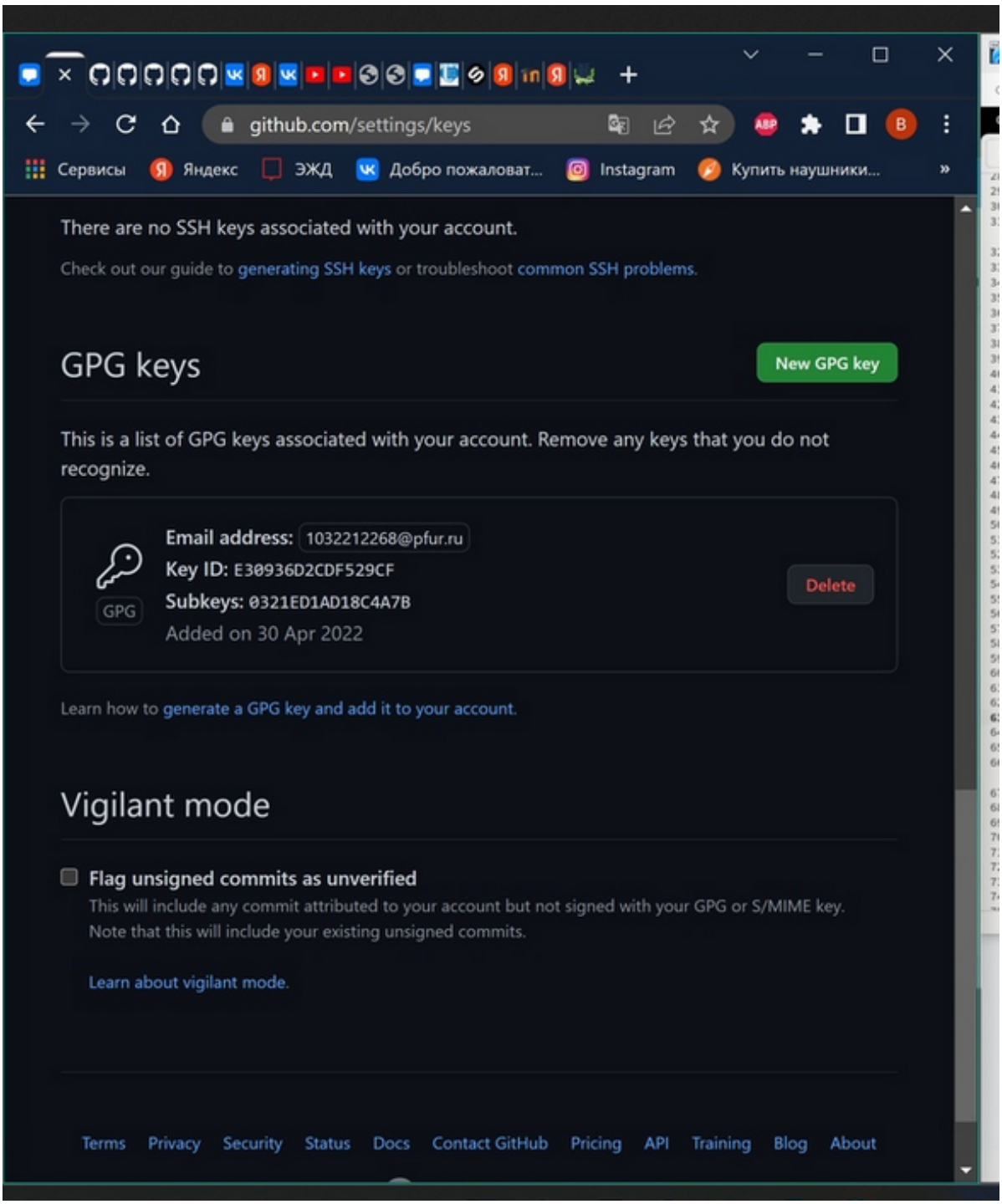


Рис. 0.2: регистрация

2. Создаём ключи.


[vvduplenskikh@vvduplenskikh ~]$ gpg --list-secret-keys --keyid-format LONG
/home/vvduplenskikh/.gnupg/pubring.kbx
---
sec rsa4096/FAEA12ACBF768F71 2022-04-25 [SC]
FE18ED05714DF8DB5D765AD1FAEA12ACBF768F71
uid [абсолютно] Vasily <1032212268@pfur.ru>
ssb rsa4096/DDE818C78CC06CB0 2022-04-25 [E]
sec rsa4096/2FFA42F559462D20 2022-04-29 [SC]
B679A86075B0E20F8EF606E02FFA42F559462D20
uid [абсолютно] Vasily <1032212268@pfur.ru>
ssb rsa4096/9D95E6BC2E97C75E 2022-04-29 [E]
sec rsa4096/A62AFD175C4792B9 2022-04-30 [SC]
37C0576473799050D9B61B00A62AFD175C4792B9
uid [абсолютно] Vasily Duplenskih <1032212268@pfur.ru>
ssb rsa4096/20FAF4CFD50CB397 2022-04-30 [E]
sec rsa4096/E30936D2CDF529CF 2022-04-30 [SC] [годен до: 2032-04-27]
1C44EFB743DC90BFF5FF57FBE30936D2CDF529CF
uid [абсолютно] Vasily Douplenskih <1032212268@pfur.ru>
ssb rsa4096/0321ED1AD18C4A7B 2022-04-30 [E] [годен до: 2032-04-27]
[vvduplenskikh@vvduplenskikh ~]$



3. Настройка автоматических подписей коммитов git.

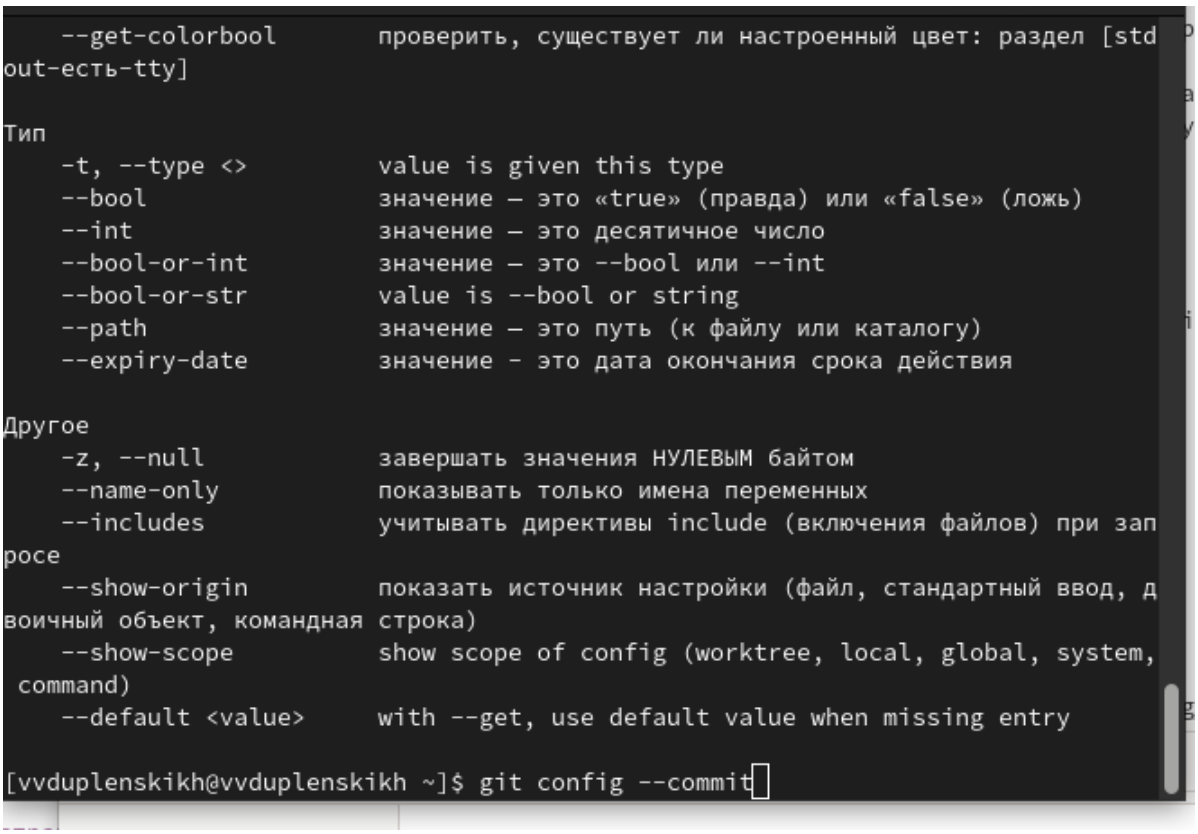


Рис. 0.3: Подписи

4. Настройка gh.

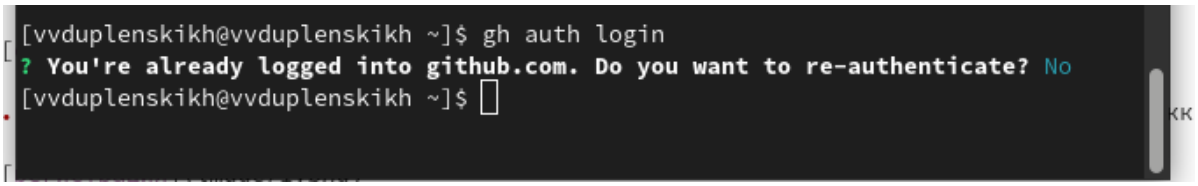


Рис. 0.4: gh

5. Подготовка репозитория и коммит изменений.

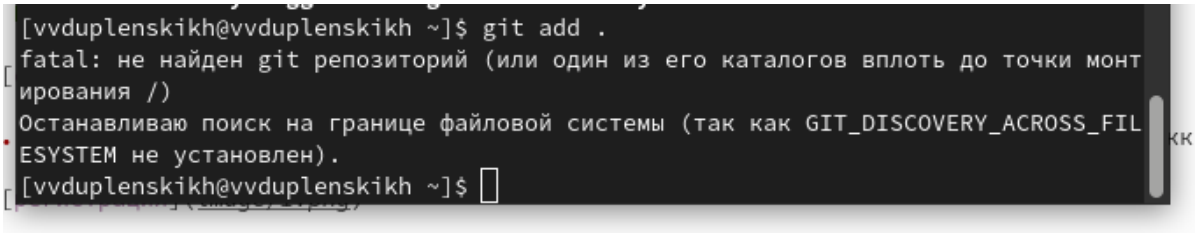


Рис. 0.5: Подготовка репозитория

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

- хранилище - пространство на накопителе где расположен репозиторий
- commit - сохранение состояния хранилища
- история - список изменений хранилища (коммитов)
- рабочая копия - локальная копия сетевого репозитория, в которой работает программист. 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 осуществлялся 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”.

4. Опишите действия с VCS при единоличной работе с хранилищем.

Один пользователь работает над проектом и по мере необходимости делает коммиты, сохраняя определенные этапы.

5. Опишите порядок работы с общим хранилищем VCS.

Несколько пользователей работают каждый над своей частью проекта. При этом каждый должен работать в своей ветки. При завершении работы ветка пользователя сливается с основной веткой проекта.

6. Каковы основные задачи, решаемые инструментальным средством git?

- Ведение истории версий проекта: журнал (log), метки (tags), ветвления (branches).
- Работа с изменениями: выявление (diff), слияние (patch, merge).

- Обеспечение совместной работы: получение версии с сервера, загрузка обновлений на сервер.

7. Назовите и дайте краткую характеристику командам git.

- git config - установка параметров
- git status - полный список изменений файлов, ожидающих коммита
- git add . - сделать все измененные файлы готовыми для коммита.
- git commit -m “[descriptive message]” - записать изменения с заданным сообщением.
- git branch - список всех локальных веток в текущей директории.
- git checkout [branch-name] - переключиться на указанную ветку и обновить рабочую директорию.
- git merge [branch] — соединить изменения в текущей ветке с изменениями из заданной.
- git push - запушить текущую ветку в удаленную ветку.
- git pull - загрузить историю и изменения удаленной ветки и произвести слияние с текущей веткой.

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

- git remote add [имя] [url] — добавляет удалённый репозиторий с заданным именем;
- git remote remove [имя] — удаляет удалённый репозиторий с заданным именем;
- git remote rename [старое имя] [новое имя] — переименовывает удалённый репозиторий;
- git remote set-url [имя] [url] — присваивает репозиторию с именем новый адрес;
- git remote show [имя] — показывает информацию о репозитории.

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

Ветвление — это возможность работать над разными версиями проекта: вместо одного списка с упорядоченными коммитами история будет расходиться в определённых точках. Каждая ветвь содержит легковесный указатель HEAD на последний коммит, что позволяет без лишних затрат создать много веток. Ветка по умолчанию называется master, но лучше назвать её в соответствии с разрабатываемой в ней функциональностью.

10. Как и зачем можно игнорировать некоторые файлы при commit?

Нет проблем если шаблон для игнорирования подходит для файла под контролем версий, или вы добавили файл, который игнорируется. Шаблоны не имеют никакого эффекта на файлы под контролем версий, они только определяют показываются неизвестные файлы, или просто игнорируются. Файл git.rignore обычно должен быть под контролем версий, что бы новые копии ветки видели такие же шаблоны: git add . gitignore git commit -m “Добавлены шаблоны для игнорирования”. Многие деревья с исходным кодом содержат файлы, которые не нужно хранить под контролем версий, например, резервные файлы текстового редактора, объектные файлы и собранные программы. Вы можете просто не добавлять их, но они всегда будут обнаруживаться как неизвестные. Вы также можете сказать bzr игнорировать их добавив их в файл в корне рабочего дерева. Этот файл содержит список шаблонов файлов, по одному в каждой строчке. Обычное содержимое может быть таким: .o ~ .tmp .py [co] Если шаблон содержит слеш, то он будет сопоставлен с полным путем начиная от корня рабочего дерева; иначе он сопоставляется только с именем файла. Таким образом пример выше игнорирует файлы с расширением .o во всех подкаталогах, но пример ниже игнорирует только config.h в корне рабочего дерева и HTML файлы в каталоге doc/: ./config.h doc/.html Для получения списка файлов которые игнорируются и соответствующих им шаблонов используйте команду git ignored : $ git ignored config.h ./config.h configure.in~ ~ $

Выводы

Я приобрел практические навыки работы с сервисом github.