Oleynikgit / labs Public

Code Issues Pull requests Actions Projects Wiki Security Insights Se

main

labs / lab3 / report.md

Oleynikgit Create report.md

1 contributor

86 lines (70 sloc) | 18 KB

Отчет по лабораторной работе № 2

управление версиями git

Олейник Артём Александрович

Содержание

1 Цель работы

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

2 Задание

-1. Зарегистрироваться на Github;
-2. Создать базовую конфигурацию для работы с git;
-3. Создать ключ SSH;

```
-4. Создать ключ PGP;
-5.  Настроить подписи git;
-6. Создать локальный каталог для выполнения заданий по предмету.
```

3 Теоретическое введение

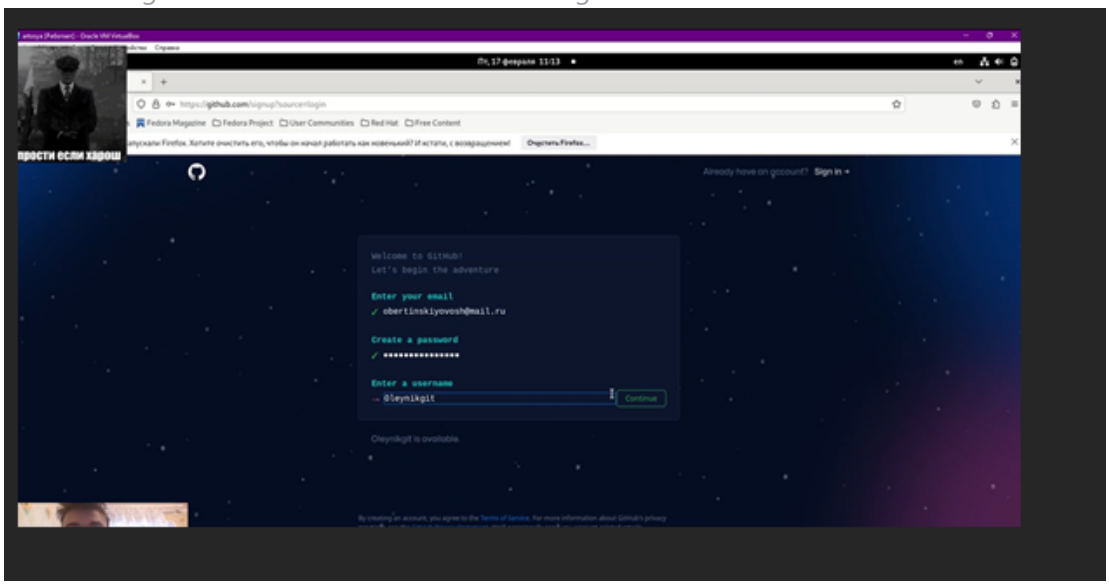
В этой лабораторной работе мы познакомимся с системами контроля версий. Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников 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. Существуют классические и распределенные системы контроля версий (РСКВ). Сегодня мы будем работать с распределенной VSC – Git. В РСКВ (таких как Git, Mercurial, Bazaar или Darcs) клиенты не просто скачивают снимок всех файлов - они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных. Более того, многие РСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах. [1]

4 Выполнение лабораторной работы

1 Создаем учетную запись на Github и заполняем основные данные. (рис. 1)

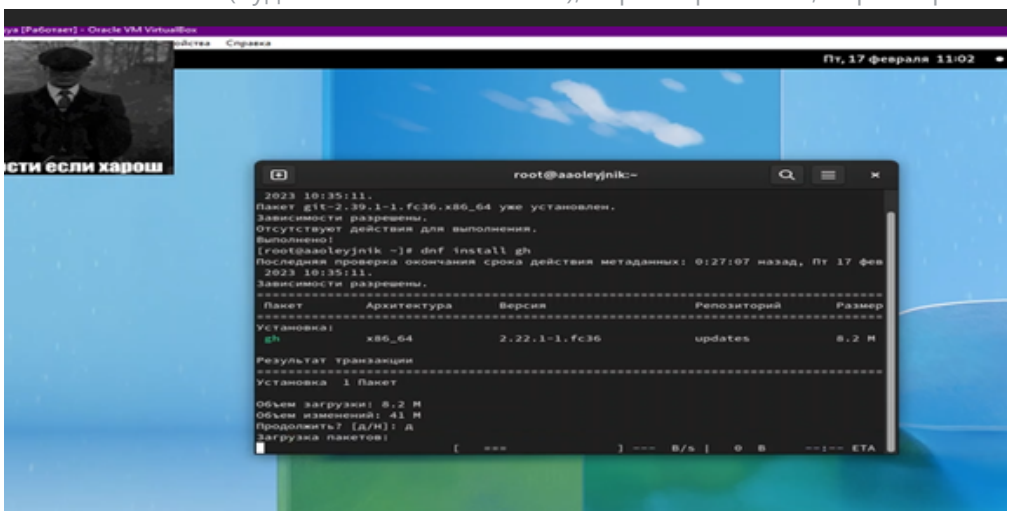
Рис. 1: Создание учетной записи на GitHub 2 Далее установим программное обеспечение git-flow в Fedora Linux (сделаем это вручную

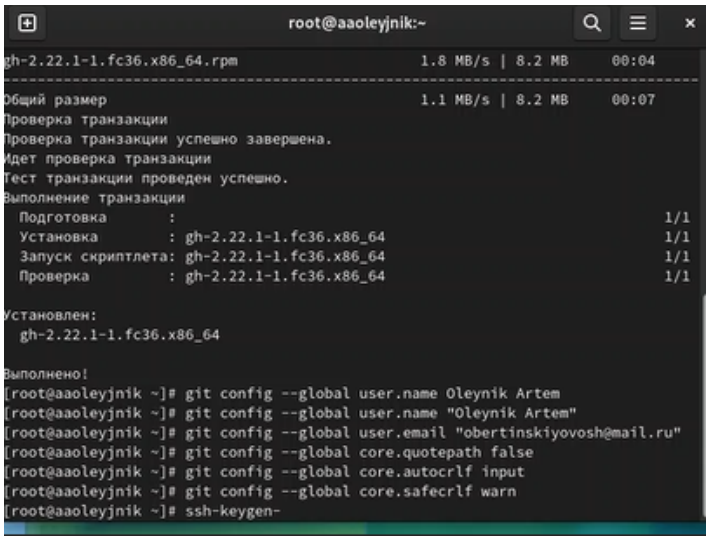
Установка git-flow в Fedora Linux Установим gh в Fedora Linux.



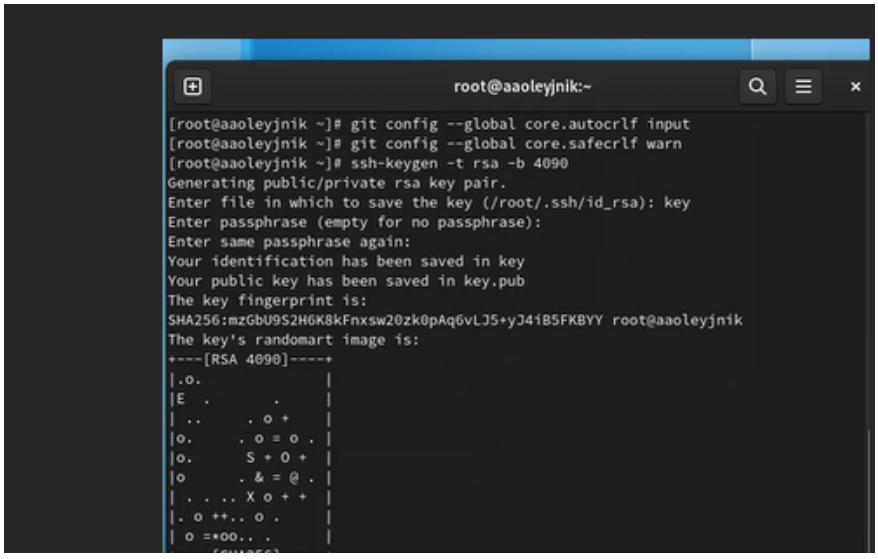
Установка gh в Fedora Linux

Перейдем к базовой настройке Git: зададим имя и почту владельца репозитория; настроим utf-8 в выводе сообщений git; настроим верификацию и подписание коммитов git (Зададим имя начальной ветки (будем называть её master), параметр autocrlf, параметр safecrlf). (рис. 4)

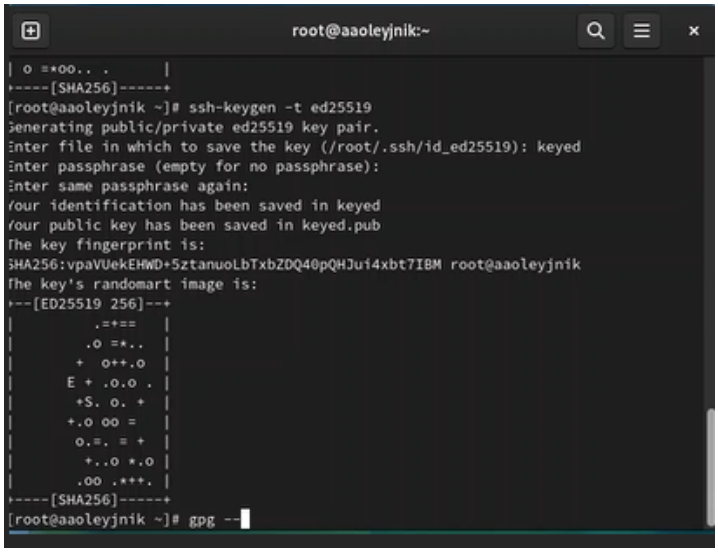




: Базовая настройка



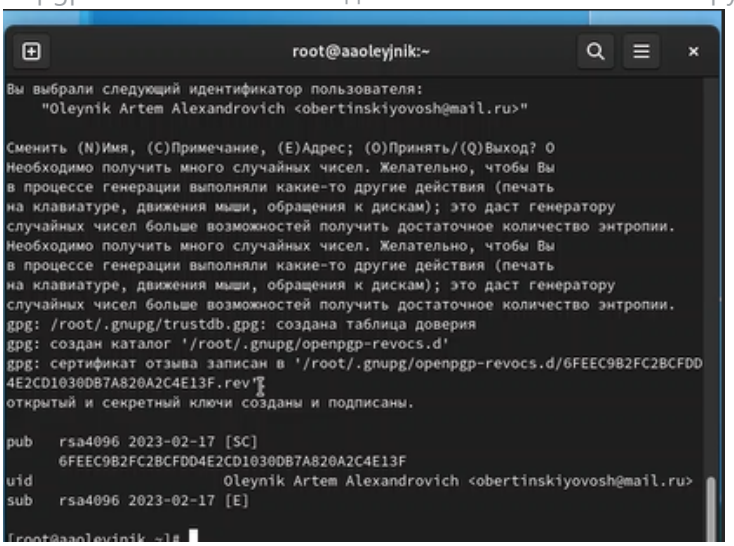
Создание ключа ssh по алгоритму rsa с ключем размером 4096 по алгоритму ed25519: Создание ключа ssh по алгоритму ed25519 4 Создаем ключ gpg.Генерируем ключ и из предложенных опций выбираем



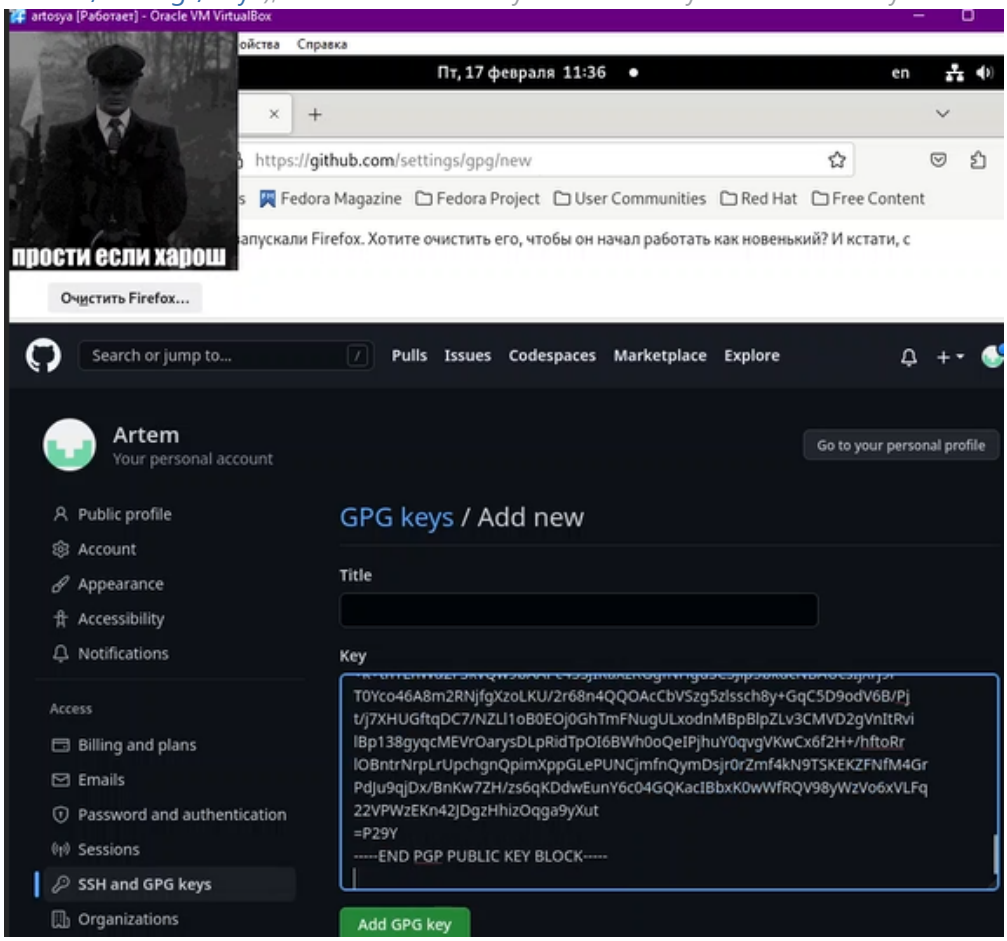
Тип RSA and RSA; • Размер 4096; • Выберите срок действия; значение по умолчанию— 0 (срок действия не истекает никогда). • GPG запросит личную информацию, которая сохранится в ключе: • Имя (не менее 5 символов). • Адрес электронной почты. • При вводе email убедитесь, что он соответствует адресу, используемому на GitHub. • Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить это поле пустым. (рис. 7)

Рис. 7: Создание ключа gpg

Добавим pgp ключ в Github.Выводим список ключей и компируем отпечаток приватного ключа

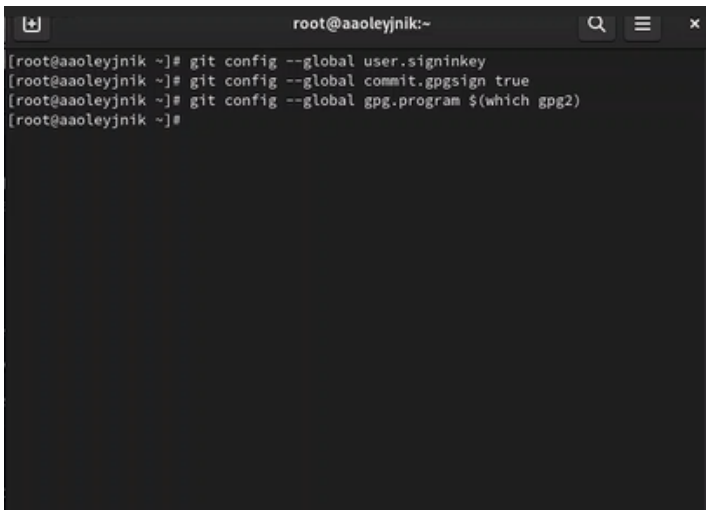


(рис.8) Перейдем в настройки GitHub (https://github.com/settings/keys), нажмем на кнопку New GPG key и вставим полученный ключ в

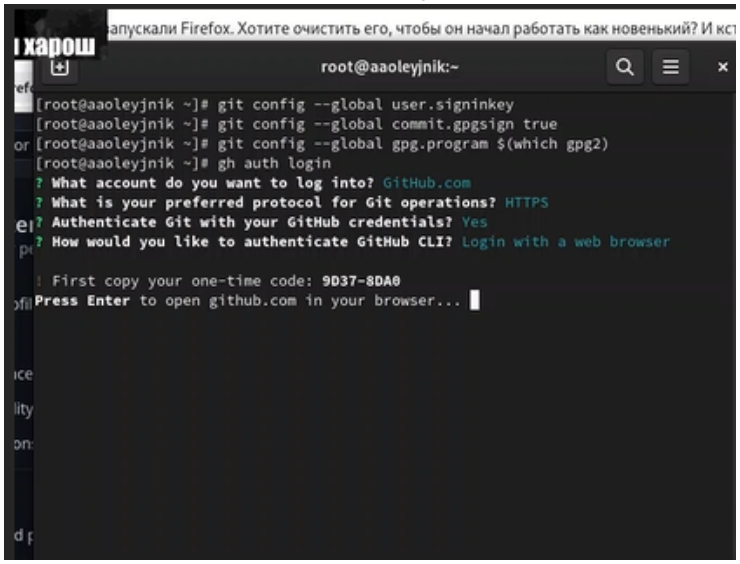


поле ввода

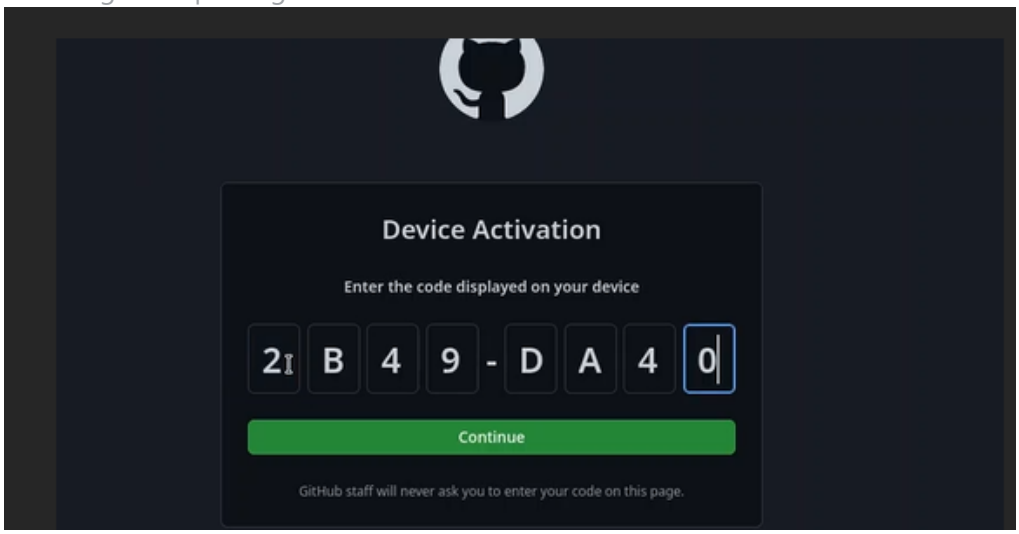
Добавление PGP ключа в GitHub 5 Настроим автоматические подписи коммитов git. Используя введёный email, укажим Git применять его при подписи коммитов. (рис. 11)



Настроика автоматических подписей коммитов git настроим gh.



Настройка gh



Для начала необходимо авторизоваться.Ответим на несколько входящих вопросов,которые задаст утилита.Авторизуемся через браузер

5 Выводы

Изучил средства контроля версий и научился применять их.Освол работу с git,научился подключать репозитории,добавлять и удалять необходимые файлы.

6 Ответы на контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? Система контроля версий — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет 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 для: • Хранение полной истории изменений • причин всех производимых изменений • Откат изменений, если что-то пошло не так • Поиск причины и ответственного за появления ошибок в программе • Совместная работа группы над одним проектом • Возможность изменять код, не мешая работе других пользователей
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Репозиторий - хранилище версий - в нем хранятся все документы вместе с историей их изменения и другой служебной информацией. Commit — отслеживание изменений, сохраняет разницу в изменениях Рабочая копия - копия проекта, связанная с репозиторием (текущее состояние файлов проекта, основанное на версии из хранилища (обычно на последней)) История хранит все изменения в проекте и позволяет при необходимости обратиться к нужным данным.
3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. Централизованные VCS (Subversion; CVS; TFS; VAULT; AccuRev): • Одно основное хранилище всего проекта • Каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет и, затем, добавляет свои изменения обратно Децентрализованные VCS (Git; Mercurial; Bazaar): • У каждого пользователя свой вариант (возможно не один) репозитория • Присутствует возможность добавлять и забирать изменения из любого репозитория [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. В отличие от классических, в распределённых системах контроля версий центральный репозиторий не является обязательным.
4. Опишите действия с VCS при единоличной работе с хранилищем. Сначала создаем и подключаем удаленный репозиторий. Затем по мере изменения проекта отправлять эти изменения на сервер.
5. Опишите порядок работы с общим хранилищем VCS. 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.
6. Каковы основные задачи, решаемые инструментальным средством git? Первая — хранить информацию о всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом.
7. Назовите и дайте краткую характеристику командам 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 с текущим 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. Приведите примеры использования при работе с локальным и удалённым репозиториями. git push –all (push origin master/любой branch)
9. Что такое и зачем могут быть нужны ветви (branches)? Ветвление («ветка», branch) — один из параллельных участков истории в одном хранилище, исходящих из одной версии (точки ветвления). [3] • Обычно есть главная ветка (master), или ствол (trunk). • Между ветками, то есть их концами, возможно слияние. Используются для разработки новых функций.
10. Как и зачем можно игнорировать некоторые файлы при commit? 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.

Список литературы

1. О системе контроля версий [Электронный ресурс]. 2016. URL: https://git-scm.com/book/ru/v2/Введение-О-системе-контроля-версий.
2. Евгений Г. Системы контроля версий [Электронный ресурс]. 2016. URL: https://glebradchenko.susu.ru/courses/bachelor/engineering/2016/SUSU_SE_2016_REP_3_VCS.pdf.

Give feedback