Лабораторная работа №2

Управление версиями

Магомед Асхабович Мажитов

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Задания

- Создать базовую конфигурацию для работы с git.
- Создать ключ SSH.
- Создать ключ PGP.
- Настроить подписи git.
- Зарегистрироваться на Github.
- Создать локальный каталог для выполнения заданий по предмету.

3 Ход работы

1. В первую очередь я зарегистрировался на гитхабе.(рис. **??**)

Регистрация на гитхабе

2. Установил gitflow.(рис. **??**)

Установка gitflow

3. Установил gh в Fedora Linux с помощью команды *sudo dnf install gh*.(рис. **??**)

Установка gh

4. Выполнил базовую настройку *git*. С помощью команд *git config –global user.name "Magomed Mazhitov"* и *git config –global user.email "magomed_mazhitov_03@mail"* задал имя и email владельца репозитория.(рис. **??**)

Базовая настройка git

Настроил utf-8 в выводе сообщений git, задал имя начальной ветки, параметр autocrlf, параметр safecrlf(рис. **??**).

Настройка utf-8, задание имени начальной ветки, параметра autocrlf и safecrlf

5. Создал ключ SSH по алгоритму rsa и ed25519(рис. **??**, **??**).

Создание ключа SSH по алгоритму rsa

Создание ключ SSH по алгоритму ed25519

6. Сгенерировал GPG ключ с помощью команды *gpg –full-generate-key*. Далее из предложенных опций выбрал все как в инструкции(рис. **??**).

Генерирование ключа Gpg

7. Вывел список ключей с помощью команды *gpg –list-secret-keys –keyid-format LONG*. Затем с помощью команды *gpg –armor –export _____ | xclip -sel clip* скопировал сгенерированный GPG ключ в буфер обмена, где вместо пропуска ввел

отпечаток ключа(рис. **??**).

Копирование GPG ключа в буфер обмена

8. Создал на гитхабе новый GPG ключ и вставил ключ из буфера обмена(рис. **??**).

Добавление GPG ключа на гитхаб

9. Настроил автоматические подписи коммитов git(рис. **??**).

Настройка автоматических подписей коммитов

10. Далее я авторизовался с помощью команды *gh auth login*(рис. **??**).

Авторизация gh

11. Создаю репозиторий на основе предаставленного шаблона(рис. **??**).

Создание репозитория

12. Настроил каталог курса. Сначала я перешел в него с помощью команды *cd*, затем удалил лишние файлы, создал необходимые каталоги(рис. **??**).

Настройка каталога курса

13. Отправил файлы на сервер(рис. **??**).

Отправка файлов на сервер

4 Вывод

Мы изучили идеологию и применение средств контроля версий, а также освоили умения по работе с git.

5 Контрольные вопросы.

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?

- Система контроля версий (VCS) - 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. Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов.

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

- Хранилище – репозиторий - место хранения всех версий и служебной информации.
- Commit - это команда для записи индексированных изменений в репозиторий.
- История – место, где сохраняются все коммиты, по которым можно посмотреть данные о коммитах.
- Рабочая копия – текущее состояние файлов проекта, основанное на версии, загруженной из хранилища.

3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида

- Централизованные системы – это системы, в которых одно основное хранилище всего проекта, и каждый пользователь копирует необходимые ему файлы, изменяет и вставляет обратно. Пример – Subversion.
- Децентрализованные системы – система, в которой каждый пользователь имеет свой вариант репозитория и есть возможность добавлять и забирать изменения из репозиториев. Пример – Git.

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

- В рабочей копии, которую исправляет человек, появляются правки, которые отправляются в хранилище на каждом из этапов. То есть в правки в рабочей копии появляются, только если человек делает их (отправляет их на сервер) и никак по-другому .

5. Опишите порядок работы с общим хранилищем VCS.

- Если хранилище общее, то в рабочую копию каждого, кто работает над проектом, приходят изменения, отправленные на сервер одним из команды. Рабочая правка каждого может изменяться вне зависимости от того, делает ли конкретный человек правки или нет.

6. Каковы основные задачи, решаемые инструментальным средством git?

- У Git две основных задачи: первая — хранить информацию обо 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.

- 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
- удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm имена_файлов
- 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: 1 git push origin имя_ветки
- слияние ветки с текущим деревом: 1 git merge —no-ff имя_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.

- Работа с удаленным репозиторием: git remote – просмотр списка настроенных удаленных репозиториев.
- Работа с локальным репозиторием: git status - выводит информацию обо всех изменениях, внесенных в дерево директорий проекта по сравнению с последним коммитом рабочей ветки

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

- Ветка (англ. branch) — это последовательность коммитов, в которой ведётся параллельная разработка какого-либо функционала. Ветки нужны, чтобы несколько программистов могли вести работу над одним и тем же проектом или даже файлом одновременно, при этом не мешая друг другу. Кроме того, ветки используются для тестирования экспериментальных функций: чтобы не повредить основному проекту, создается новая ветка специально для экспериментов.

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

- Игнорируемые файлы — это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты. В Git нет специальной команды для указания игнорируемых файлов: вместо этого необходимо вручную отредактировать файл . Временно игнорировать изменения в файле можно командой git update-index-assumeunchanged