Отчёт по лабораторной работе 1

Управление версиями

Румянцева Александра Сергеевна

Содержание

Список иллюстраций

Список таблиц

Цель работы

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

Задание

Лабораторная работа подразумевает изучение основных команд в git.

Теория

Git (произносится «гит») — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года.

Система спроектирована как набор программ, специально разработанных с учётом их использования в сценариях. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Например, Cogito является именно таким примером оболочки к репозиториям Git, а StGit использует Git для управления коллекцией исправлений (патчей).

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удалённый доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

Выполнение лабораторной работы

1. Создадим учётную запись на https://github.com (рис. 1).

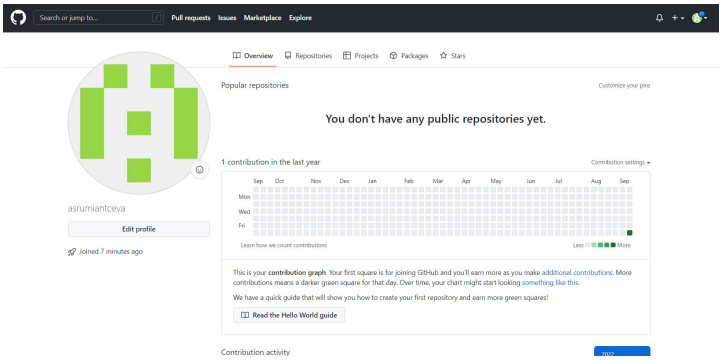

Рис. 1: рис. 1. Учетная запись на Github.

2. Проверила установку git (рис. 2)

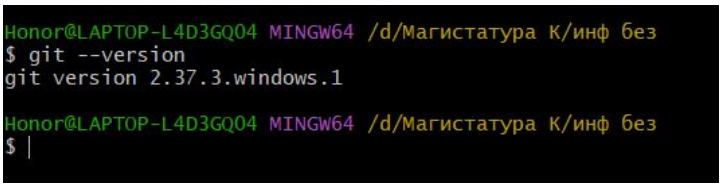

Рис. 2: рис. 2. Установленная версия Git.

3. Сделала предварительную конфигурацию, указав имя email владельца репозитория (рис 3.).

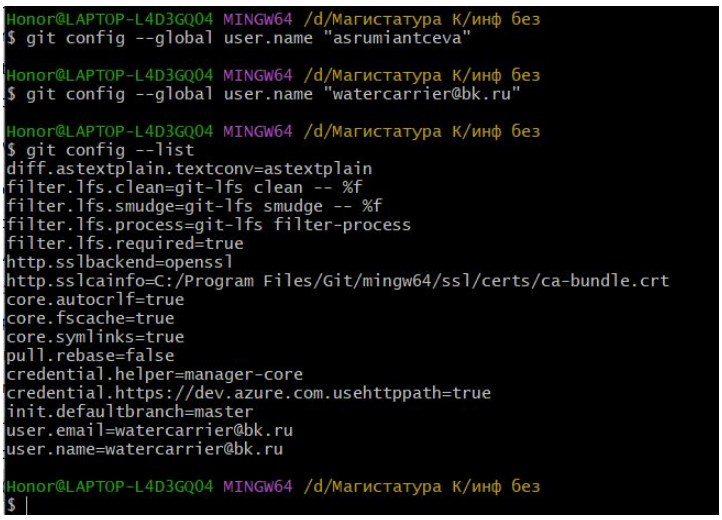

Рис. 3: рис. 3. Указание владельца репрозитория.

4. Настроим систему контроля версий git, как это указано в инструкции к 1-ой лаборатной работе с использованием сервера репозиториев https://github.com/.

Для этого необходимо сгенерировать пару ключей (приватный и открытый), а затем вставить их в SSH-ключи на github (рис. 4).

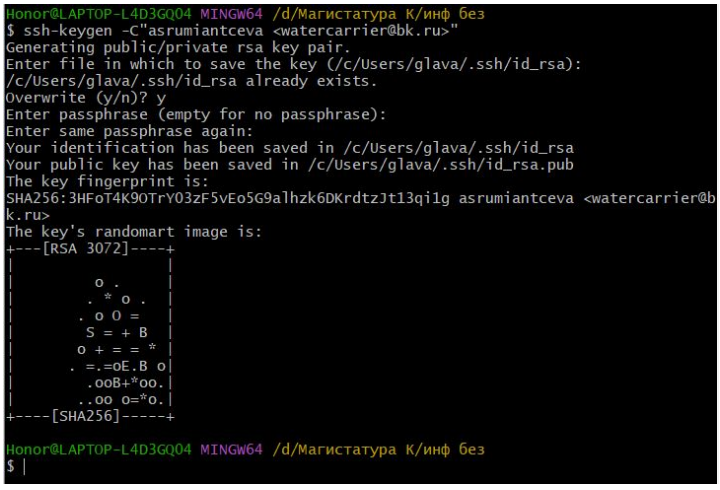

Рис. 4: рис. 4. Генерация SSH ключа.

Создание ключа gpg (рис. 5)

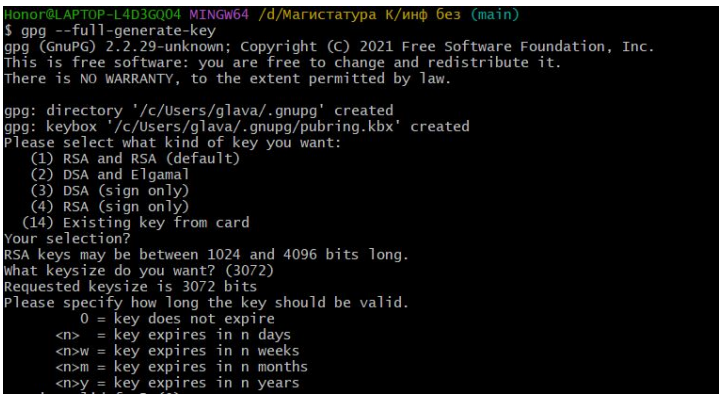

Рис. 5: рис. 5. Создание gpg ключа.

5. Следующим шагом будет создание и подключение репозитория к GitHub.

В GitHub заходим в «repository» и создаём новый репозиторий (имя «laboratory», а заголовок для файла README). Копируем в консоль ссылку на репозиторий (для дальнейшей работы с файлами):

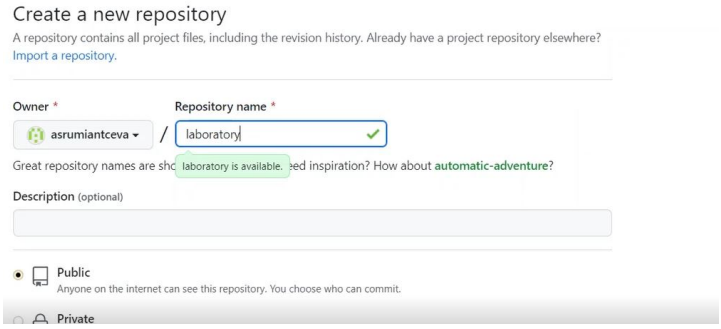

Рис. 6: рис. 6. Создание репозитория.

С помощью git добавляем файлы в репозиторий (рис. 7-9)

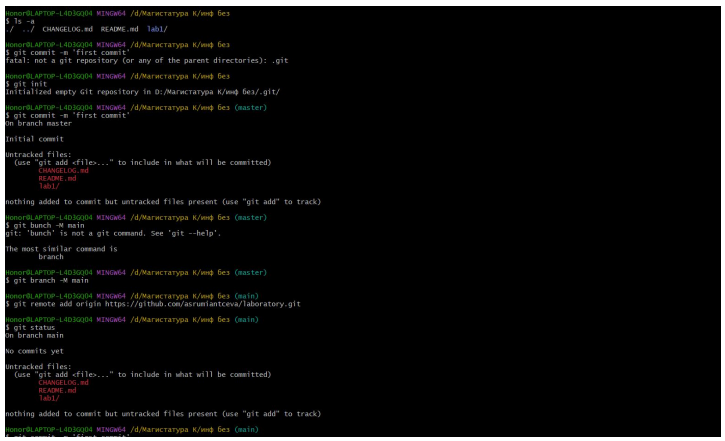

Рис. 7: рис. 7. Добавление файлов в репозиторий 1.

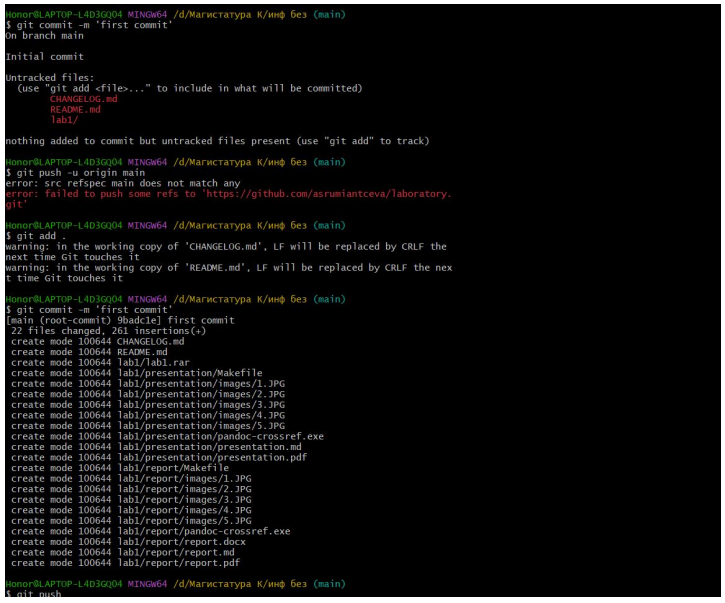

Рис. 8: рис. 8. Добавление файлов в репозиторий 2.

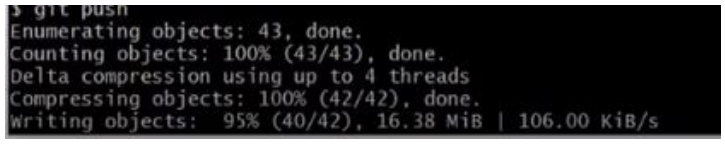

Рис. 9: рис. 9. Добавление файлов в репозиторий 3.

Файлы добавлены в репозиторий

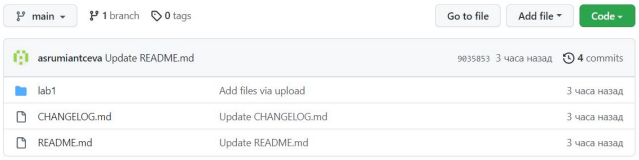

Рис. 10: рис. 10. Добавленные файлы в репозиторий.

Библиография

1. ТУИС РУДН

2. Статья о Git https://learn.microsoft.com/ru-ru/devops/develop/git/what-is-git

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.

2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище (repository), или репозитарий — место хранения всех версий и служебной информации.

Commit («[трудовой] вклад», не переводится) — синоним версии; процесс создания новой версии.

Рабочая копия (working copy) — 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”. Еще 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. Опишите действия с VCS при единоличной работе с хранилищем.

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. Опишите порядок работы с общим хранилищем VCS.

Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый):

ssh-keygen -C “Имя Фамилия 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. Каковы основные задачи, решаемые инструментальным средством git?

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. 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

– 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

– 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

– отправка изменений конкретной ветки в центральный репозиторий: git push origin имя_ветки

– слияние ветки с текущим деревом: git merge –no-ff имя_ветки

8. Приведите примеры использования при работе с локальным и удалённым репозиториями. Использования git при работе с локальными репозиториями (добавления текстового документа в локальный репозиторий):

git добавить hello.txt
git commit -am ‘Новый файл’

9. Что такое и зачем могут быть нужны ветви (branches)? Ветки нужны для того, чтобы программисты могли вести совместную работу над проектом и не мешать друг другу при этом. Кроме того, с помощью branches решаются следующие проблемы: нужно постоянно создавать архивы с рабочим кодом, сложно “переключаться” между архивами, сложно перетаскивать изменения между архивами, легко что-то напутать или потерять.

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. Во время работы над проектом эти файлы могут создаваться, но их 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 » .гитиньоре

curl -L -s https://www.gitignore.io/api/c ++ » .gitignore

Выводы

Я изучила идеологию и применение средств контроля версий. Освоила умения по работе в git.