Отчет по лабораторной работе №2

Дисциплина: Операционные системы

Оширова Юлия Николаевна, НКАбд-02-22

Содержание

# 1 Цель работы

Изучить идеологию и применение средств контроля версий. Приобрести практические навыки по работе с системой 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 учктную запись на Github и заполняем основные данные.

№2

Сделала предварительную конфигурацию git: открыла терминал и ввела следующие команды, указав имя и email владельца репозитория:

git config - -global user.name “” git config - -global user.email “[work@email](mailto:work@email)

Настроила utf-8 в выводе сообщений git: git config - -globalcore.quotepath false

Задала имя начальной ветки master: git config –globalinit.defaultBranch master

Задала параметр autocrlf: git config –global core.autocrlf input

Задала параметр safecrlf: git config –global core.safecrlf warn

![Базовая настройка Git](data:application/octet-stream;base64,)

Базовая настройка Git

![Настройка utf-8 в выводе сообщений git](data:application/octet-stream;base64,)

Настройка utf-8 в выводе сообщений git

![Задала имя начальной ветки master](data:application/octet-stream;base64,)

Задала имя начальной ветки master

![Задала параметр autocrlf и safecrlf](data:application/octet-stream;base64,)

Задала параметр autocrlf и safecrlf

№3

Сгенерировала пару ключей – приватный и открытый:

ssh-keygen -C “Имя Фамилия [work@mail](mailto:work@mail)

![Создание SSH ключа](data:application/octet-stream;base64,)

Создание SSH ключа

Скопировала из локальной консоли ключ в буфер обмена:

cat ~/.ssh/id\_rsa.pub | xclip -sel clip

![Копирование в буфер обмена](data:application/octet-stream;base64,)

Копирование в буфер обмена

Загрузила сгенерированный открытый ключ: зашла на сайт https://github.org/ под своей учетной записью и перешла в меню Settings (настройки), в боковом меню выбрали «SSH and GPG keys» и нажала кнопку «New SSH key», вставила ключ в появившемся на сайте поле и указали имя для ключа – Title:

![Загрузка сгенерированного SSH ключа](data:application/octet-stream;base64,)

Загрузка сгенерированного SSH ключа

№4

Генерируем ключ

gpg –full-generate-key

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.

![Генерация GPG ключа](data:application/octet-stream;base64,)

Генерация GPG ключа

![Выбираем тип, размер и срок действия](data:application/octet-stream;base64,)

Выбираем тип, размер и срок действия

Выводим список ключей и копируем отпечаток приватного ключа:

gpg –list-secret-keys –keyid-format LONG

Отпечаток ключа — это последовательность байтов, используемая для идентификации более длинного, по сравнению с самим отпечатком ключа.

Формат строки:

sec Алгоритм/Отпечаток\_ключа Дата\_создания [Флаги] [Годен\_до]ID\_ключа

Экспортируем ключ в формате ASCII по его отпечатку:

gpg –armor –export

![Выводим список ключей и копируем отпечаток приватного ключа](data:application/octet-stream;base64,)

Выводим список ключей и копируем отпечаток приватного ключа

![Экспортируем ключ в формате ASCII по его отпечатку](data:application/octet-stream;base64,)

Экспортируем ключ в формате ASCII по его отпечатку

***Добавление PGP ключа в GitHub***

Копируем ключ и добавляем его в настройках профиля на GitHub (или GitLab).

Cкопируйте ваш сгенерированный PGP ключ в буфер обмена:

gpg –armor –export | xclip -sel clip

Перейдите в настройки GitHub (https://github.com/settings/keys), нажмите на кнопку New GPG key и вставьте полученный ключ в поле ввода.

![Вставляем полученный ключ в поле ввода](data:application/octet-stream;base64,)

Вставляем полученный ключ в поле ввода

№5

***Настройка автоматических подписей коммитов git***

Используя введёный email, укажите Git применять его при подписи коммитов:

git config –global user.signingkey git config –global commit.gpgsign true git config –global gpg.program $(which gpg2)

![Настройка автоматических подписей коммитов git](data:application/octet-stream;base64,)

Настройка автоматических подписей коммитов git

***Настройка gh***

Для начала необходимо авторизоваться

gh auth login

Утилита задаст несколько наводящих вопросов. Авторизоваться можно через броузер.

![Настройка gh](data:application/octet-stream;base64,)

Настройка gh

№6

***Создание репозитория курса на основе шаблона***

Необходимо создать шаблон рабочего пространства (см. Рабочее пространство для лабораторной работы).

Например, для 2022–2023 учебного года и предмета «Операционные системы» (код предмета os-intro) создание репозитория примет следующий вид:

mkdir -p ~/work/study/2022-2023/"Операционные системы"  
cd ~/work/study/2022-2023/"Операционные системы"  
gh repo create study\_2022-2023\_os-intro --template=yamadharma/course-directory-student-template --public  
git clone --recursive git@github.com:<owner>/study\_2022-2023\_os-intro.git os-intro

![Создание репозитория курса на основе шаблона](data:application/octet-stream;base64,)

Создание репозитория курса на основе шаблона

***Настройка каталога курса***

Перейдите в каталог курса: cd ~/work/study/2022-2023/“Операционные системы”/os-intro

Удалите лишние файлы: rm package.json

Создайте необходимые каталоги:

echo os-intro > COURSE make

Отправьте файлы на сервер:

git add . git commit -am ‘feat(main): make course structure’ git push

![Настройка каталога курса](data:application/octet-stream;base64,) ![Настройка каталога курса](data:application/octet-stream;base64,)

# 5 Выводы

Изучила идеологию и применение средств контроля версий. Приобрели практические навыки по работе с системой git.

# 6 Контрольные вопросы для самопроверки

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