Лабораторная работа №2

Дисциплина - операционные системы

Волгин Иван Алексеевич

Содержание

# 1 Цель работы

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

# 2 Задание

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 Теоретическое введение

### 3.0.1 Системы контроля версий. Общие понятия

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.

В отличие от классических, в распределённых системах контроля версий центральный репозиторий не является обязательным.

Среди классических VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд.

### 3.0.2 Примеры использования git

Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды git с различными опциями.  
Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией.

# 4 Выполнение лабораторной работы

Для начала устанавливаем git и gh с помощью команд dnf install git, dnf install gh (рис. fig. 1).

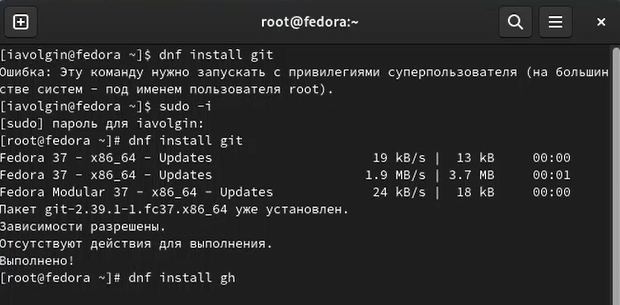


Рис. 1: Установка git и gh

Далее нам нужно задать имя (git config –global user.name “Name Surname”) и email (git config –global user.email “work@mail”) владельца репозитория и настроить utf-8 в выводе сообщений git (git config –global core.quotepath false) (рис. fig. 2).

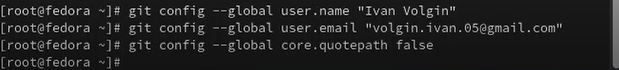


Рис. 2: Задаем имя и email владельца репозитория и настраиваем utf-8

Затем задаем имя начальной ветки (будем называть её master) (git config –global init.defaultBranch master), параметр autocrlf (git config –global core.autocrlf input) и параметр safecrlf (git config –global core.safecrlf warn) (рис. fig. 3).

Задаем имя начальной ветки и параметры autocrlf и safecrlf.

Рис. 3: Задаем имя начальной ветки и параметры autocrlf и safecrlf.

Создаем ключи ssh. Превый по алгоритму rsa с ключом размером 4096 бит (ssh-keygen -t rsa -b 4096) (рис. fig. 4), и второй по алгоритму ed25519 (ssh-keygen -t ed25519) (рис. fig. 5).

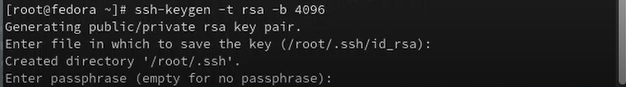


Рис. 4: Создаем ключ ssh по алгоритму rsa.

Создаем ssh ключ по алгоритму ed25519.

Рис. 5: Создаем ssh ключ по алгоритму ed25519.

Далее генерируем ключ pgp (gpg –full-generate-key) и из предложенных опций выбираем тип RSA and RSA, размер 4096 и бесконечный срок действия (рис. fig. 6).

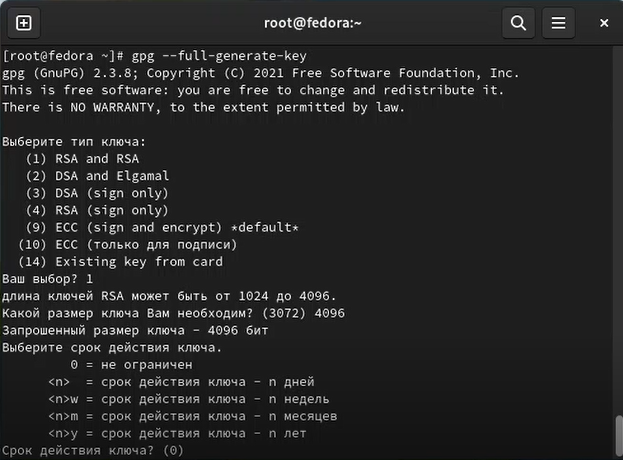


Рис. 6: Создание ключа pgp.

Далее GPG запросит информацию, которая будет храниться в ключе: имя, адрес электронной почты и комментарий(рис. fig. 7).

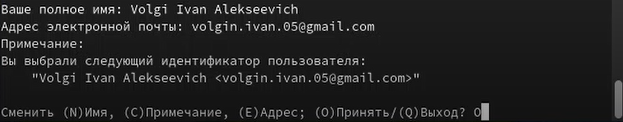


Рис. 7: Личная информация, запрошенная GPG.

Далее нам нужно добавить PGP ключ в GitHub. Для этого выводим список ключей в терминал (gpg –list-secret-keys –keyid-format LONG) (рис. fig. 8) и копируем PGP ключ в буфер обмена (gpg –armor –export | xclip -sel clip) (рис. fig. 9). Затем переходим в настройки GitHub, нажимаем кнопку New PGP key и вставляем полученный ключ в поле ввода (рис. fig. 10).

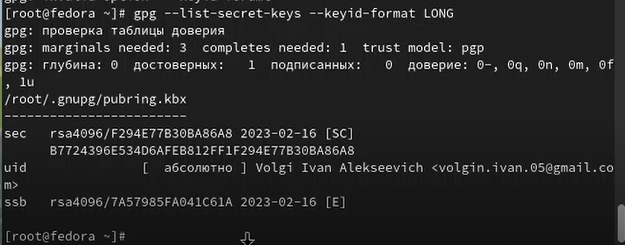


Рис. 8: Выводим список ключей в терминал.

Копируем ключ PGP в буфер обмена.

Рис. 9: Копируем ключ PGP в буфер обмена.

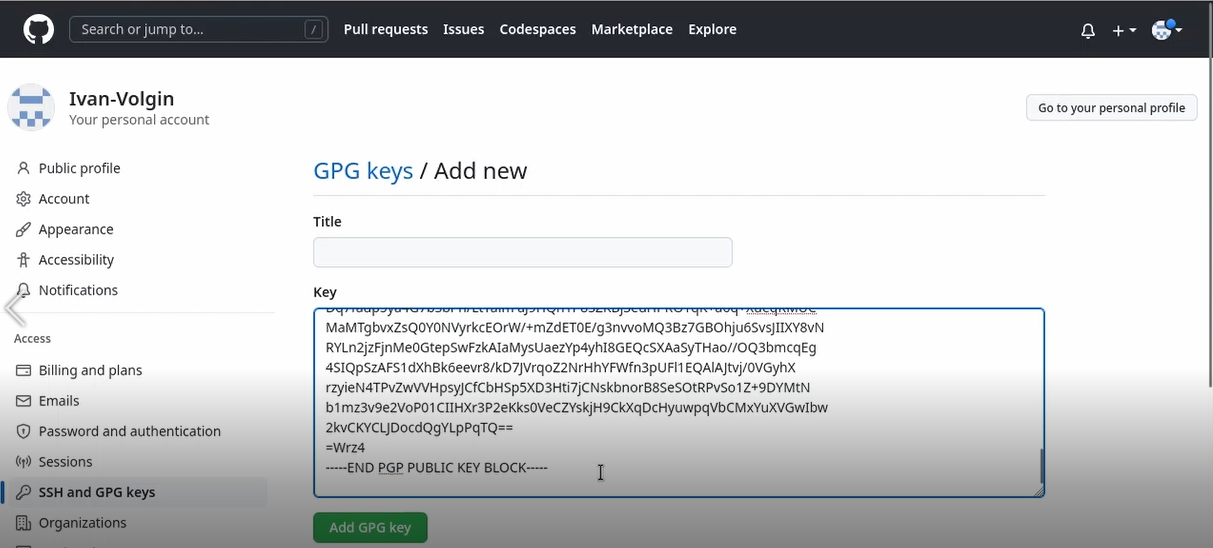


Рис. 10: Вставляем PGP ключ на GitHub.

атем настраиваем автоматические подписи коммитов git (рис. fig. 11). git config –global user.signingkey git config –global commit.gpgsign true git config –global gpg.program $(which gpg2)

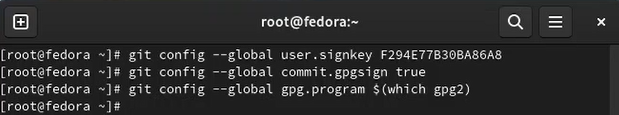


Рис. 11: Настройка автоматических подписей git.

Далее мы должны настроить gh. Для начала авторизовываемся (gh auth login) (рис. fig. 12). Затем система задает несколько наводящих вопросов и авторизуемся через броузер (рис. fig. 13).

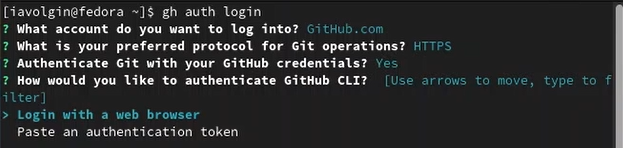


Рис. 12: Авторизуемся и отвечаем на вопросы.

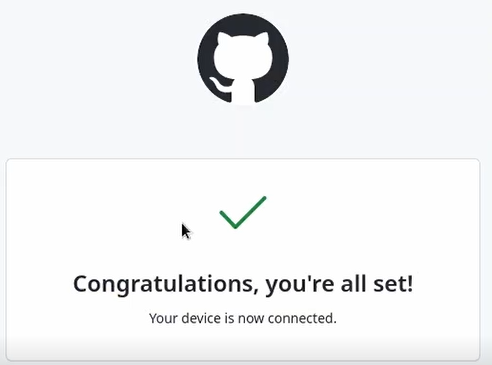


Рис. 13: Авторизуемся через броузер.

После этого создаём репозиторий курса на основе шаблона. mkdir -p ~/work/study/2022-2023/“Операционные системы” (создаем в файловой системе директорию) (рис. fig. 14). cd ~/work/study/2022-2023/“Операционные системы” (переходим в файл “Операционные системы”) (рис. fig. 14).

gh repo create study\_2022-2023\_os-intro –template=yamadharma/course-directory-student-template –public (создаем на гитхабе репозиторий на основе шаблона) (рис. fig. 15). git clone –recursive git@github.com:/study\_2022-2023\_os-intro.git os-intro (клонируем репозиторий себе в систему) (рис. fig. 16).

Создаем директорию и переходим в нужный файл.

Рис. 14: Создаем директорию и переходим в нужный файл.

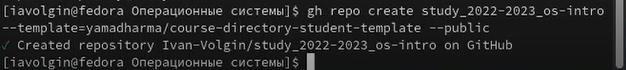


Рис. 15: Создаем на гитхабе репозиторий на основе шаблона.

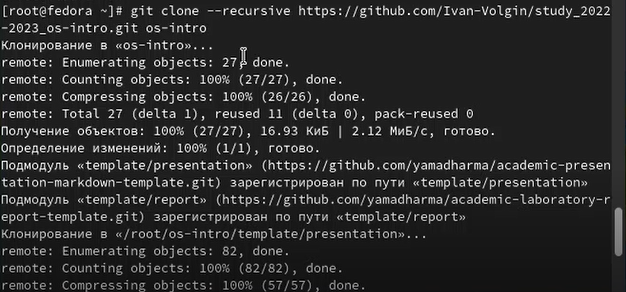


Рис. 16: Клонируем репозиторий себе в систему.

Далее настраиваем каталог курса. Переходим в каталог (cd ~/work/study/2022-2023/“Операционные системы”/os-intro), удаляем лишние файлы (rm package.json), создаем необходимые каталоги (echo os-intro > COURSE , make) и отправляем все фалы на сервер (git add . , git commit -am ‘feat(main): make course structure’ , git push). Часть видео с выполнением этих команд у меня не записалась.

# 5 Выводы

В ходе выполнения лабораторной работы я изучил идеологию и применение средств контроля версий, так же освоил умения по работе с git.

# 6 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? Система управления версиями (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. Применяются они для хранения полной истории изменений, совместной работы команды над одним проектом, хранения полной информации о каждом изменении (кто и когда).
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище – репозиторий, в котором хранятся все документы, история их изменений и прочая информация. Commit – отслеживание изменений, сохраняет разницу в изменениях. История – хранилище всех изменений, позволяющее в любой момент вернуться к прежней версии проекта. Рабочая копия – копия проекта, основанная на версии из хранилища, зачастую последней.
3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. Централизованные VCS – одно основное хранилище проекта, из которого каждый пользователь может брать себе для работы нужную копию файлов и после их изменения возвращать обратно. Децентрализованные VCS – личный вариант репозитория каждого пользователя, есть возможность забирать и добавлять изменения из любого репозитория.
4. Опишите действия с VCS при единоличной работе с хранилищем. Сначала подключается удаленный репозиторий. Затем вносятся изменения и отправляются обратно на сервер.
5. Опишите порядок работы с общим хранилищем VCS. Пользователь перед началом работы должен взять нужную ему версию проекта из хранилища, затем внести изменения и отправить их обратно на сервер, тем самым создав новую версию проекта. Старые версии тоже сохранятся.
6. Каковы основные задачи, решаемые инструментальным средством git? Хранение всех версий проекта, истории изменений и упрощение командной работы.
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 изменений: • 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

1. Приведите примеры использования при работе с локальным и удалённым репозиториями. При работе с локальным репозиторием нам может понадобиться получить обновления из центрального репозитория, тогда мы будем должны использовать команду git pull. Аналогично, чтобы внести уже проработанные изменения в удаленный репозиторий мы должны использовать команду git push.
2. Что такое и зачем могут быть нужны ветви (branches)? Каждая ветвь представляет собой полную копию материнской ветви, но изменения которые на ней происходят не отображаются на материнской ветви ( потом можно провести слияние). Они нужны для удобства работы разных разработчиков или отделов разработки над одним проектом. Так они не мешают друг другу.
3. Как и зачем можно игнорировать некоторые файлы при commit? При работе могут создаваться фалы, которые не нужно отправлять в удаленный репозиторий. Чтобы избежать их попадания туда можно добавить шаблоны инорируемых при добавлении в репозиторий типов файлов.