Лабароторный работа номер 2

Отчет пределаных работ

Студент: Джахангиров.и.з

Группа: НКАбд-05-22

**МОСКВА**

**2023**

**Цели лабораторной работы**

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

Основные команды git

* Перечислил наиболее часто используемые команды git.
* Создал основного дерева репозитория:
* git init
* Получил обновлений (изменений) текущего дерева из центрального репозитория:
* git pull
* Отправил всех произведённых изменений локального дерева в центральный репозиторий:
* git push
* Просмотрил списка изменённых файлов в текущей директории:
* git status
* Просмотрил текущих изменений:
* git diff
* Сохранение текущих изменений:
  + добавил все изменённые и/или созданные файлы и/или каталоги:
  + git add .
  + добавил конкретные изменённые и/или созданные файлы и/или каталоги:
  + git add имена\_файлов
  + удалил файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории):
  + git rm имена\_файлов
* Сохранение добавленных изменений:
  + сохранил все добавленные изменения и все изменённые файлы:
  + git commit -am 'Описание коммита'
  + сохранил добавленные изменения с внесением комментария через встроенный редактор:
  + git commit
  + создал 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)
  + отправил изменений конкретной ветки в центральный репозиторий:
  + git push origin имя\_ветки
  + слияние ветки с текущим деревом:
  + git merge --no-ff имя\_ветки
* Удаление ветки:
  + удалил локальной уже слитой с основным деревом ветки:
  + git branch -d имя\_ветки
  + принудительное удалил локальной ветки:
  + git branch -D имя\_ветки
  + удаление ветки с центрального репозитория:
  + git push origin :имя\_ветки

Стандартные процедуры работы при наличии центрального репозитория

* Работа пользователя со своей веткой начинается с проверки и получения изменений из центрального репозитория (при этом в локальное дерево до начала этой процедуры не должно было вноситься изменений):
* git checkout master
* git pull
* git checkout -b имя\_ветки
* Затем можно вносить изменения в локальном дереве и/или ветке.
* После завершения внесения какого-то изменения в файлы и/или каталоги проекта необходимо разместить их в центральном репозитории. Для этого необходимо проверить, какие файлы изменились к текущему моменту:
* git status
* При необходимости удалил лишние файлы, которые не хотим отправлять в центральный репозиторий.
* Затем полезно просмотреть текст изменений на предмет соответствия правилам ведения чистых коммитов:
* git diff
* Если какие-либо файлы не должны попасть в коммит, то помечаем только те файлы, изменения которых нужно сохранить. Для этого используем команды добавления и/или удаления с нужными опциями:
* git add …
* git rm …
* Если нужно сохранить все изменения в текущем каталоге, то используем:
* git add .
* Затем сохранил изменения, поясняя, что было сделано:
* git commit -am "Some commit message"
* Отправил изменения в центральный репозиторий:
* git push origin имя\_ветки

или

git push

### Работа с локальным репозиторием

* 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"
* Настроим 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
* После это в каталоге tutorial появится каталог .git, в котором будет храниться история изменений.
* Создадим тестовый текстовый файл hello.txt и добавим его в локальный репозиторий:
* echo 'hello world' > hello.txt
* git add hello.txt
* git commit -am 'Новый файл'
* Воспользувался командой status для просмотра изменений в рабочем каталоге, сделанных с момента последней ревизии:
* git status
* 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. 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 >> .gitignore
* curl -L -s https://www.gitignore.io/api/c++ >> .gitignore

### Работа с сервером репозиториев

* Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый):
* ssh-keygen -C "Имя Фамилия <work@mail>"
* Ключи сохранился в каталоге ~/.ssh/.
* Существует несколько доступных серверов репозиториев с возможностью бесплатного размещения данных. Например, <https://github.com/>.
* Для работы с ним необходимо сначала завести на сайте <https://github.com/> учётную запись. Затем необходимо загрузить сгенерённый нами ранее открытый ключ.
* Для этого зайти на сайт <https://github.com/> под своей учётной записью и перейти в меню GitHub setting.
* После этого выбрал в боковом меню GitHub setting>SSH-ключи и нажать кнопку Добавить ключ. Скопировав из локальной консоли ключ в буфер обмена:
* cat ~/.ssh/id\_rsa.pub | xclip -sel clip
* Вставляем ключ в появившееся на сайте поле.
* После этого я создал на сайте репозиторий, выбрав в меню , дать ему название и сделать общедоступным (публичным).
* Для загрузки репозитория из локального каталога на сервер выполняем следующие команды:
* git remote add origin
* ssh://git@github.com/<username>/<reponame>.git
* git push -u origin master
* Далее на локальном компьютере можно выполнять стандартные процедуры для работы с git при наличии центрального репозитория.

## Базовая настройка git

### Первичная настройка параметров git

* Задал имя и email владельца репозитория:
* git config --global user.name "Name Surname"
* git config --global user.email "work@mail"
* Настроим utf-8 в выводе сообщений git:
* git config --global core.quotepath false
* Настройте верификацию и подписание коммитов git.
* Задал имя начальной ветки (будем называть её master):

git config --global init.defaultBranch master

## Создал ключа ssh

### Общая информация

1. Алгоритмы шифрования ssh
   1. Аутентификация

В SSH поддерживается четыре алгоритма аутентификации по открытым ключам:

* + - DSA:
      * размер ключей DSA не может превышать 1024, его следует отключить;
    - RSA:
      * следует создавать ключ большого размера: 4096 бит;
    - ECDSA:
      * ECDSA завязан на технологиях NIST, его следует отключить;
    - Ed25519:
      * используется пока не везде.
  1. Симметричные шифры
     + Из 15 поддерживаемых в SSH алгоритмов симметричного шифрования, безопасными можно считать:
       - chacha20-poly1305;
       - aes\*-ctr;
       - aes\*-gcm.
     + Шифры 3des-cbc и arcfour потенциально уязвимы в силу использования DES и RC4.
     + Шифр cast128-cbc применяет слишком короткий размер блока (64 бит).
  2. Обмен ключами
     + Применяемые в SSH методы обмена ключей DH (Diffie-Hellman) и ECDH (Elliptic Curve Diffie-Hellman) можно считать безопасными.
     + Из 8 поддерживаемых в SSH протоколов обмена ключами вызывают подозрения три, основанные на рекомендациях NIST:
       - ecdh-sha2-nistp256;
       - ecdh-sha2-nistp384;
       - ecdh-sha2-nistp521.
     + Не стоит использовать протоколы, основанные на SHA1.

1. Файлы ssh-ключей
   1. По умолчанию пользовательские ssh-ключи сохраняются в каталоге ~/.ssh в домашнем каталоге пользователя.
   2. Убедил, что у меня ещё нет ключа.
   3. Файлы закрытых ключей имеют названия типа id\_<алгоритм> (например, id\_dsa, id\_rsa).
      * По умолчанию закрытые ключи имеют имена:
      * id\_dsa
      * id\_ecdsa
      * id\_ed25519
      * id\_rsa
   4. Открытые ключи имеют дополнительные расширения .pub.
      * По умолчанию публичные ключи имеют имена:
      * id\_dsa.pub
      * id\_ecdsa.pub
      * id\_ed25519.pub
      * id\_rsa.pub
   5. При создании ключа команда попросит ввести любую ключевую фразу для более надёжной защиты вашего пароля. Можно пропустить этот этап, нажав Enter.
   6. Сменить пароль на ключ можно с помощью команды:
   7. ssh-keygen -p

### Создание ключа ssh

* Ключ ssh создаём командой:
* ssh-keygen -t <алгоритм>
* Создайте ключи:
  + по алгоритму rsa с ключём размером 4096 бит:
  + ssh-keygen -t rsa -b 4096
  + по алгоритму ed25519:
  + ssh-keygen -t ed25519
* При создании ключа команда попросит ввести любую ключевую фразу для более надёжной защиты вашего пароля. Можно пропустить этот этап, нажав Enter.
* Сменить пароль на ключ можно с помощью команды:
* ssh-keygen -p

### Добавил SSH-ключа в учётную запись GitHub

* Скопируйте созданный SSH-ключ в буфер обмена командой:
* xclip -i < ~/.ssh/id\_ed25519.pub
* Откройте настройки своего аккаунта на GitHub и перейдем в раздел SSH and GPC keys.
* Нажмите кнопку ew SSH key.
* Добавьте в поле Title название этого ключа, например, ed25519@hostname.
* Вставьте из буфера обмена в поле Key ключ.
* Нажмите кнопку Add SSH key.

## Верификация коммитов с помощью PGP

* Как настроить PGP-подпись коммитов с помощью gpg.

### Общая информация

* Коммиты имеют следующие свойства:
  + author (автор) — контрибьютор, выполнивший работу (указывается для справки);
  + committer (коммитер) — пользователь, который закоммитил изменения.
* Эти свойства можно переопределить при совершении коммита.
* Авторство коммита можно подделать.
* В git есть функция подписи коммитов.
* Для подписывания коммитов используется технология PGP (см. [Работа с PGP](https://esystem.rudn.ru/mod/notes/public/20201218123900-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0_%D1%81_pgp.md)).
* Подпись коммита позволяет удостовериться в том, кто является коммитером. Авторство не проверяется.

### Создание ключа

* 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.

### Экспорт ключа

* 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 <PGP Fingerprint>

### 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 <PGP Fingerprint> | 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.

### Подписывание коммитов git

* Подпись коммитов при работе через терминал:
* git commit -a -S -m 'your commit message'
* Флаг -S означает создание подписанного коммита. При этом может потребоваться ввод кодовой фразы, заданной при генерации GPG-ключа.

### Настройка автоматических подписей коммитов git

* Использувал введёный email, укажите Git применять его при подписи коммитов:
* git config --global user.signingkey <PGP Fingerprint>
* git config --global commit.gpgsign true
* git config --global gpg.program $(which gpg2)

## Проверка коммитов в Git

* GitHub и GitLab будут показывать значок Verified рядом с вашими новыми коммитами.

### Режим бдительности (vigilant mode)

* На GitHub есть настройка [vigilant mode](https://docs.github.com/en/github/authenticating-to-github/managing-commit-signature-verification/displaying-verification-statuses-for-all-of-your-commits).
* Все неподписанные коммиты будут явно помечены как Unverified.
* Включается это в настройках в разделе SSH and GPG keys. Установите метку на Flag unsigned commits as unverified.

# **Последовательность выполнения работы**

## Установил программного обеспечения

### Установка git

* Установим git:
* dnf install git

### Установка gh

* Fedora:
* dnf install gh

## Базовая настройка git

* Зададим имя и email владельца репозитория:
* git config --global user.name "Name Surname"
* git config --global user.email "work@mail"
* Настроим utf-8 в выводе сообщений git:
* git config --global core.quotepath false
* Настройте верификацию и подписание коммитов git (см. [Верификация коммитов git с помощью GPG](https://esystem.rudn.ru/mod/notes/public/20210128184700-%D0%B2%D0%B5%D1%80%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82%D0%BE%D0%B2_git_%D1%81_%D0%BF%D0%BE%D0%BC%D0%BE%D1%89%D1%8C%D1%8E_gpg.md)).
* Зададим имя начальной ветки (будем называть её master):
* git config --global init.defaultBranch master
* Параметр autocrlf:
* git config --global core.autocrlf input
* Параметр safecrlf:
* git config --global core.safecrlf warn

## Создали ключи ssh

* по алгоритму rsa с ключём размером 4096 бит:
* ssh-keygen -t rsa -b 4096
* по алгоритму ed25519:
* ssh-keygen -t ed25519

## Создали ключи pgp

* 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.

## Настройка github

* Создали учётную запись на [https://github.com](https://github.com/).
* Заполнил основные данные на [https://github.com](https://github.com/).

## Добавление PGP ключа в GitHub

* 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
* Cкопируйте ваш сгенерированный PGP ключ в буфер обмена:
* gpg --armor --export <PGP Fingerprint> | 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.

## Настройка автоматических подписей коммитов git

* Используя введёный email, укажите Git применять его при подписи коммитов:
* git config --global user.signingkey <PGP Fingerprint>
* git config --global commit.gpgsign true
* git config --global gpg.program $(which gpg2)

## Настройка gh

* Для начала необходимо авторизоваться
* gh auth login
* Утилита задаст несколько наводящих вопросов.
* Авторизоваться можно через броузер.

## Шаблон для рабочего пространства

* [Рабочее пространство для лабораторной работы](https://esystem.rudn.ru/mod/notes/public/20210116125100-%D1%80%D0%B0%D0%B1%D0%BE%D1%87%D0%B5%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%B4%D0%BB%D1%8F_%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B8_%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B.md)
* Репозиторий: <https://github.com/yamadharma/course-directory-student-template>.

### Сознание репозитория курса на основе шаблона

* Необходимо создал шаблон рабочего пространства (см. [Рабочее пространство для лабораторной работы](https://esystem.rudn.ru/mod/notes/public/20210116125100-%D1%80%D0%B0%D0%B1%D0%BE%D1%87%D0%B5%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%B4%D0%BB%D1%8F_%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B8_%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B.md)).
* 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

### 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

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.
3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.
4. Опишите действия с VCS при единоличной работе с хранилищем.
5. Опишите порядок работы с общим хранилищем VCS.
6. Каковы основные задачи, решаемые инструментальным средством git?
7. Назовите и дайте краткую характеристику командам git.
8. Приведите примеры использования при работе с локальным и удалённым репозиториями.
9. Что такое и зачем могут быть нужны ветви (branches)?
10. Как и зачем можно игнорировать некоторые файлы при commit?
11. Создал базовую конфигурацию для работы с git.
12. Создал ключ *SSH*.
13. Создал ключ *PGP*.
14. Настроилподписи git.
15. Зарегистрировался на *Github*.
16. Создал локальный каталог для выполнения заданий по предмету.