

Отчет по лабораторной работе №2

Дисциплина: Операционные системы

Ардеев Никита Евгеньевич НММбд-01-23

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	11
5	Контрольные вопросы	12

Список иллюстраций

3.1	Установка git	7
3.2	Базовая настройка	7
3.3	Создание ssh ключа	8
3.4	Генерация ключей	8
3.5	Настройка автоматических подписей коммитов git	9

Список таблиц

1 Цель работы

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

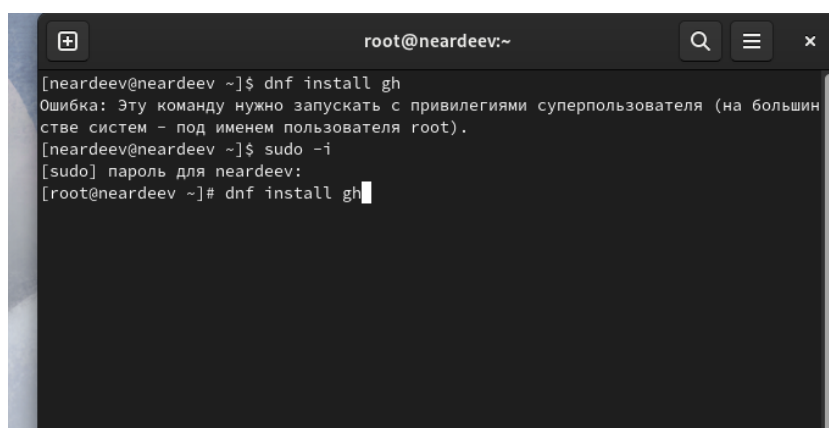
2 Задание

1. Создать базовую конфигурацию для работы с git.
2. Создать ключ SSH.
3. Создать ключ PGP.
4. Настроить подписи git.
5. Зарегистрироваться на Github.
6. Создать локальный каталог для выполнения заданий по предмету.

3 Выполнение лабораторной работы

Установил git: (рис. 3.1).

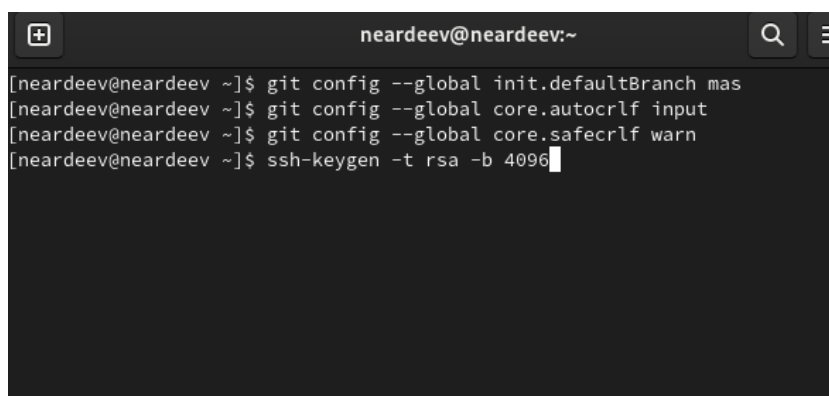
\$ dnf install git



```
root@neardeev:~  
[neardeev@neardeev ~]$ dnf install gh  
Ошибка: Эту команду нужно запускать с привилегиями суперпользователя (на большин  
стве систем - под именем пользователя root).  
[neardeev@neardeev ~]$ sudo -i  
[sudo] пароль для neardeev:  
[root@neardeev ~]# dnf install gh
```

Рис. 3.1: Установка git

Авторизовался на github. Настроил utf-8 в выводе сообщений git. Задал имя начальной ветки, параметр autocrlf, параметр safecrlf (рис. 3.2).



```
neardeev@neardeev:~  
[neardeev@neardeev ~]$ git config --global init.defaultBranch mas  
[neardeev@neardeev ~]$ git config --global core.autocrlf input  
[neardeev@neardeev ~]$ git config --global core.safecrlf warn  
[neardeev@neardeev ~]$ ssh-keygen -t rsa -b 4096
```

Рис. 3.2: Базовая настройка

Создал ключ ssh (рис. 3.3).

```
neardeev@neardeev:~  
| .o oo.+oBX0+B|  
+----[SHA256]-----+  
[neardeev@neardeev ~]$ ssh-keygen -t ed25519  
Generating public/private ed25519 key pair.  
Enter file in which to save the key (/home/neardeev/.ssh/id_ed25519):  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/neardeev/.ssh/id_ed25519  
Your public key has been saved in /home/neardeev/.ssh/id_ed25519.pub  
The key fingerprint is:  
SHA256:I6iwJN5+5boRa3QU20gdWeP6E7qMFF/iIda4gpRUVeM neardeev@neardeev  
The key's randomart image is:  
+--[ED25519 256]--+
```

Рис. 3.3: Создание ssh ключа

Генерирую ключи gpg, создал и настроил их (рис. 3.4).

```
Не найдена команда: install. Воспользуйтесь /usr/bin/dnf --help  
Это, возможно, команда подключаемого модуля DNF, попробуйте: «dnf install 'dnf-c  
ommand(install)'»  
[root@neardeev ~]# gpg --full-generate-key  
gpg (GnuPG) 2.4.0; Copyright (C) 2021 Free Software Foundation, Inc.  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law.  
  
gpg: создан каталог '/root/.gnupg'  
gpg: создан щит с ключами '/root/.gnupg/pubring.kbx'  
Выберите тип ключа:  
(1) RSA and RSA  
(2) DSA and Elgamal  
(3) DSA (sign only)  
(4) RSA (sign only)  
(9) ECC (sign and encrypt) *default*  
(10) ECC (только для подписи)  
(14) Existing key from card  
Ваш выбор? 
```

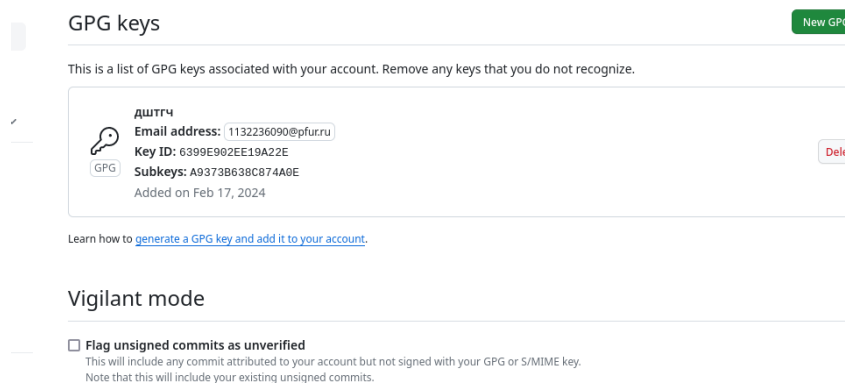
Рис. 3.4: Генерация ключей

Вывел список ключей и скопировал их в буфер обмена (рис. ??).

\$ gpg --list-secret-keys --keyid-format LONG \$ gpg --armor --export | xclip -sel clip

```
+ E .  
... O .  
O. O.O.*S= O  
+OO.O =O*.* .  
...O =OO + O  
..+.O . .  
..OO. O  
-----[SHA256]-----+  
neardeev@neardeev ~]$ gpg --list-secret-keys --keyid-format LONG
```


Добавил ключ на github(рис. ??).



Используя введённый email, указал Git применять его при подписи коммитов (рис. 3.5).

```
[neardeev@neardeev ~]$ git config --global user.signinkey 6399E902EE19A22E
[neardeev@neardeev ~]$ git config --global commit.gpgsign true
[neardeev@neardeev ~]$ git config --global gpg.program $(which gpg2)
[neardeev@neardeev ~]$
```

Рис. 3.5: Настройка автоматических подписей коммитов git

Авторизовался на github (рис. ??).

```
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? SSH
? Upload your SSH public key to your GitHub account? Skip
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: 36B2-CBD5
Press Enter to open github.com in your browser...
✓ Authentication complete.
- gh config set -h github.com git_protocol ssh
✓ Configured git protocol
✓ Logged in as neardeev
[neardeev@neardeev ~]$
[neardeev@neardeev ~]$
```

Создал репозиторий на основе шаблона курса (рис. ??).

```

- gh config set -h github.com git_protocol ssh
✓ Configured git protocol
✓ Logged in as neardeev
[neardeev@neardeev ~]$
[neardeev@neardeev ~]$ mkdir -p ~/work/study/2023-2024/"Операционные системы"
[neardeev@neardeev ~]$ cd ~/work/study/2023-2024/"Операционные системы"
[neardeev@neardeev Операционные системы]$ gh repo create study_2023-2024_os-
o --template=yamadharma/course-directory-student-template --public

```

```

[neardeev@neardeev ~]$ cat ~/.ssh/id_rsa.pub | xclip -sel clip
[neardeev@neardeev ~]$ git clone --recursive git@github.com:neardeev/study_20
2024_os-intro.git os-intro
Клонирование в «os-intro»...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 32 (delta 1), reused 18 (delta 0), pack-reused 0
Получение объектов: 100% (32/32), 18.59 КиБ | 4.65 Миб/с, готово.
Определение изменений: 100% (1/1), готово.
Подмодуль «template/presentation» (https://github.com/yamadharma/academic-presentation-markdown-template.git) зарегистрирован по пути «template/presentation»

```

Затем настроил каталог курса, а изменения загрузил на github

4 Выводы

Я освоил умения по работе с git, а именно создавать репозиторий, настраивать ключи безопасности.

5 Контрольные вопросы

1. Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. 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.
2. Хранилище версий - в нем хранятся все документы вместе с историей их изменения и другой служебной информацией. Рабочая копия - копия проекта, связанная с репозиторием. Рабочую копию необходимо периодически синхронизировать с репозиторием, эта операция предполагает отправку в него изменений, которые пользователь внес в свою рабочую копию, такая операция называется commit.
3. Централизованные системы контроля версий представляют собой приложения типа клиент-сервер, когда репозиторий проекта существует в единственном экземпляре и хранится на сервере. Доступ к нему осуществляется через специальное клиентское приложение. В качестве примеров таких программных продуктов можно привести CVS, Subversion. Децентрализованные VCS у каждого пользователя свой вариант (возможно не один) репозитория, присутствует возможность добавлять и забирать изменения из любого репозитория, примеры: Git, Mercurial, Bazaar.
4. Инициализация (создание) репозитория. Добавление новых файлов. Коммит. Любые операции с файлами (добавление, удаление или изменение).

5. Команда `git add` добавляет содержимое рабочего каталога в индекс (staging area) для последующего коммита. По умолчанию `git commit` использует лишь этот индекс, так что вы можете использовать `git add` для сборки слепка вашего следующего коммита. 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`. 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'`.
- 6.
7. Клонирование репозитория: Для начала работы над проектом вы можете клонировать удаленный репозиторий на свой локальный компьютер с помощью команды `git clone`, где - это ссылка на ваш удаленный репозиторий на GitHub.
 2. Добавление изменений и коммит: После внесения изменений в проект (например, добавление нового файла или обновление существующего), вы добавляете эти изменения в индекс с помощью `git add` и фиксируете их с комментарием коммита с помощью `git commit -m "Ваш комментарий здесь"`.
8. Отправка изменений на удаленный репозиторий: После фиксации изменений в локальном репозитории, вы отправляете их на удаленный репозиторий с помощью `git push origin`, где - это название ветки, на которой вы работаете.
9. Получение изменений из удаленного репозитория: Чтобы обновить свой

локальный репозиторий данными из удаленного репозитория, используйте `git pull origin`.

10. Создание новой ветки и слияние изменений: Для разработки новой функциональности часто создают новую ветку с помощью `git checkout -b` и затем сливают её изменения с основной веткой с помощью `git merge`.
11. Ветви (branches) в системах контроля версий, таких как Git, представляют собой “копии” основного кодового потока (обычно называемого мастер-веткой или `main`-веткой), в которых вы можете работать независимо от основного кода. Изоляция функциональности: Создание ветки позволяет изолировать новую функциональность, исправление ошибок или эксперименты от основного кода. Это предотвращает преждевременное влияние изменений на основной код. Параллельное развитие: Команде разработчиков можно работать над несколькими задачами одновременно, используя разные ветви. Это способствует параллельному развитию различных функций.
12. Игнорирование определенных файлов при коммите в Git осуществляется с помощью файла `.gitignore`. Этот файл содержит шаблоны файлов и директорий, которые Git должен игнорировать.

Вот как вы можете использовать `.gitignore` и зачем это может быть полезно:

1. Скрытие конфиденциальной информации: Если у вас есть файлы с конфиденциальной информацией, такие как пароли, ключи API или личные данные, вы можете добавить эти файлы в `.gitignore`, чтобы они не попали в репозиторий.
2. Исключение временных файлов и компиляционных результатов: Часто в проектах создаются временные файлы или файлы, сгенерированные в процессе компиляции. Их можно исключить из коммитов, чтобы не захламлять репозиторий.

3. Игнорирование локальных конфигурационных файлов: Локальные файлы настройки разработчика, которые могут меняться на каждой машине (например, файлы `.env`), также можно исключить из репозитория.
4. Избавление от кэшированных и временных файлов: Игнорирование файлов кэша или временных файлов, которые создаются в процессе работы системы, помогает избежать ненужных конфликтов при слиянии и уменьшает размер репозитория.

Чтобы использовать `.gitignore`, создайте файл с таким именем в корневой директории вашего репозитория и добавьте в него шаблоны файлов и директорий, которые нужно игнорировать. После добавления `.gitignore` выполните коммит, чтобы применить его изменения. Это поможет избежать нежелательных файлов в вашем репозитории и облегчит работу с командой разработчиков. 9. При единоличной работе с системой управления версиями (VCS) процесс может быть более простым, чем при коллективной разработке, но все равно имеет свои особенности. Вот типичные действия, которые могут выполняться при единоличной работе с репозиторием:

1. Инициализация репозитория: В начале проекта создается локальный репозиторий с помощью команды `git init`. Это действие выполняется один раз, чтобы начать отслеживание изменений.
2. Добавление файлов и коммит изменений: После внесения изменений в файлы проекта, вы добавляете их в индекс с помощью `git add` и фиксируете с изменениями с комментарием коммита через `git commit -m "Ваш комментарий"`.
3. Просмотр истории коммитов: Для просмотра истории коммитов и изменений в проекте используйте `git log`, чтобы отслеживать процесс развития проекта.

4. Создание и переключение веток: При необходимости работы над различными аспектами проекта может потребоваться создание и переключение веток с помощью `git branch` и `git checkout`.
5. Откат изменений: В случае необходимости откатить изменения до определенной точки, можно использовать `git reset` или `git revert`.
6. Работа с удаленным репозиторием (необязательно): При желании можно связать локальный репозиторий с удаленным (например, на GitHub) и отправлять изменения на удаленный сервер через `git push`.

Хотя при единоличной работе с репозиторием процесс более простой по сравнению с командной разработкой, использование системы управления версиями все равно целесообразно. Он поможет отслеживать изменения, восстанавливать предыдущие версии проекта и обеспечивать целостность и безопасность кода.

10. Работа с общим хранилищем системы управления версиями (VCS) включает в себя совместную разработку проекта несколькими участниками. Вот основные шаги и порядок работы при коллективном использовании VCS:

1. Клонирование репозитория: Каждый участник проекта клонирует общий репозиторий на свой компьютер с помощью команды `git clone`. Это создает локальную копию репозитория на каждом устройстве разработчика.
2. Создание и переключение веток: Разработчики могут создавать собственные ветки для работы над конкретными задачами с помощью `git branch` и переключаться между ветками с использованием `git checkout`.
3. Добавление и фиксация изменений: Каждый участник вносит свои изменения в проект, добавляет их в индекс с помощью `git add` и фиксирует с комментарием коммита через `git commit -m "Ваш комментарий"`. Также важно регулярно обновлять свою локальную версию через `git pull`.

4. Работа с конфликтами: В случае возникновения конфликтов при слиянии изменений, разработчики должны их разрешить вручную, обсудить изменения и продолжить слияние.
5. Отправка изменений на удаленный репозиторий: После завершения работы над задачей участник отправляет свои изменения на общий репозиторий с помощью `git push origin`.
6. Code review: Разработчики могут проводить взаимные code review, чтобы проверить и просмотреть изменения, улучшить качество кода и обменяться мнениями.
7. Интеграция изменений: Когда работа всех участников завершена, происходит слияние изменений (merge) в основную ветку проекта.
8. Устранение ошибок и поддержка: После интеграции изменений важно отслеживать ошибки, проводить тестирование и поддерживать проект в актуальном состоянии.

Это основной порядок работы с общим хранилищем VCS при коллективной разработке проекта. Соблюдение данного процесса помогает управлять изменениями, избежать конфликтов и обеспечить согласованность и качество кодовой базы проекта.