Отчет по лабораторной работе №2

Операционные системы

Зоригоо Номун

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Цель данной лабораторной работы – изучение идеологии и применения средств контроля версий, освоение умения по работе с git.

2 Задание

1. Создать базовую конфигурацию для работы с git
2. Создать ключ SSH
3. Создать ключ GPG
4. Настроить подписи Git
5. Заргеистрироваться на GitHub
6. Создать локальный каталог для выполнения заданий по предмету.

3 Выполнение лабораторной работы

3.1 Установка программного обеспечения

Устанавливаю необходимое программное обеспечение git и gh через терминал с помощью команд: dnf install git и dnf install gh (рис. 3.1).

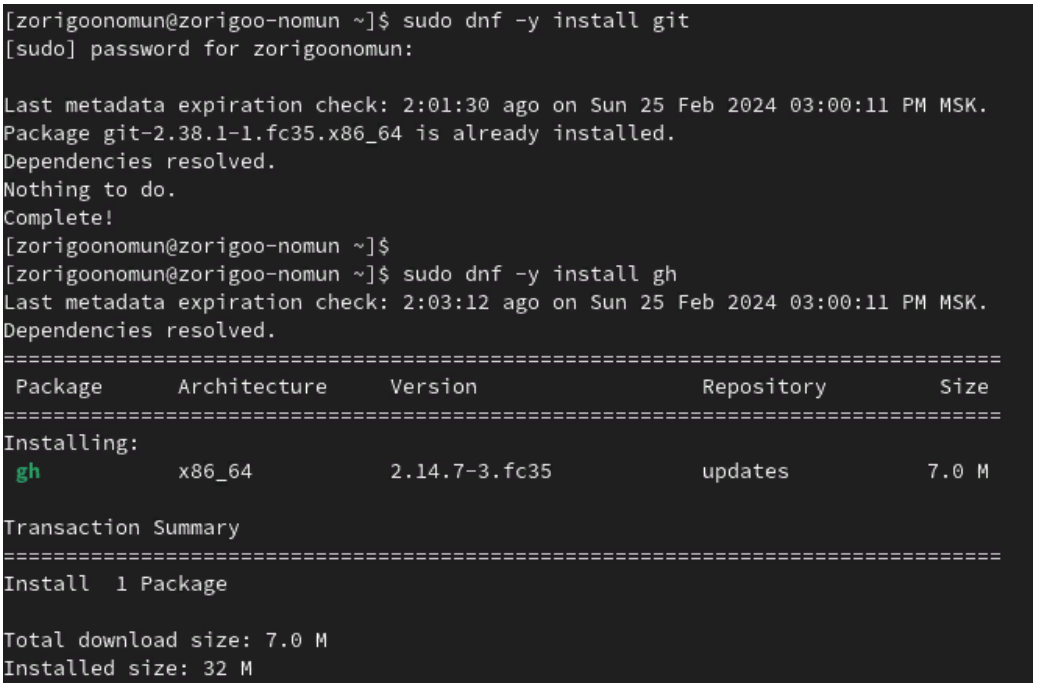

Рис. 3.1: Установка git и gh

3.2 Базовая настройка git

Задаю в качестве имени и email владельца репозитория свои имя, фамилию и электронную почту (рис. 3.2).

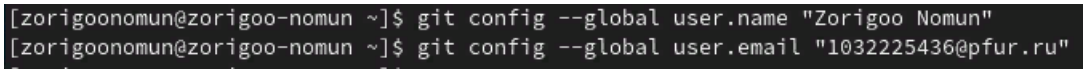

Рис. 3.2: Задаю имя и email владельца репозитория

Настраиваю utf-8 в выводе сообщений git для их корректного отображения (рис. 3.3).

```
[zorigoonomun@zorigoo-nomun ~]$ git config --global core.quotepath false
```

Рис. 3.3: Настройка utf-8 в выводе сообщений git Начальной ветке

задаю имя master (рис. 3.4).

```
zorigoonomun@zorigoo-nomun ~]$ git config --global init.defaultBranch master
```

Рис. 3.4: Задаю имя начальной ветки

Задаю параметры autocrlf и safecrlf для корректного отображения конца строки (рис. 3.5).

```
[zorigoonomun@zorigoo-nomun ~]$ git config --global core.autocrlf input
[zorigoonomun@zorigoo-nomun ~]$ git config --global core.safecrlf warn
```

Рис. 3.5: Задаю параметры autocrlf и safecrlf

3.3 Создание ключа SSH

Создаю ключ ssh размером 4096 бит по алгоритму rsa (рис. 3.6).

```
[zorigoonomun@zorigoo-nomun ~]$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/zorigoonomun/.ssh/id_rsa):
Created directory '/home/zorigoonomun/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/zorigoonomun/.ssh/id_rsa
Your public key has been saved in /home/zorigoonomun/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:FIlixehgeh7cDxGV37Lk5tBIjm3cH4MYE5Za+DFM8K4 zorigoonomun@zorigoo-nomun
The key's randomart image is:
+---[RSA 4096]----+
|    o@++..       |
|  o *.% ..       |
| + = B.=..       |
|. + =.=.+ .      |
| o . O.XS+       |
|  . ..X B o      |
|    E. + . o     |
|        . .      |
|                 |
+----[SHA256]-----+
```

Рис. 3.6: Генерация ssh ключа по алгоритму rsa

Создаю ключ ssh по алгоритму ed25519 (рис. 3.7).

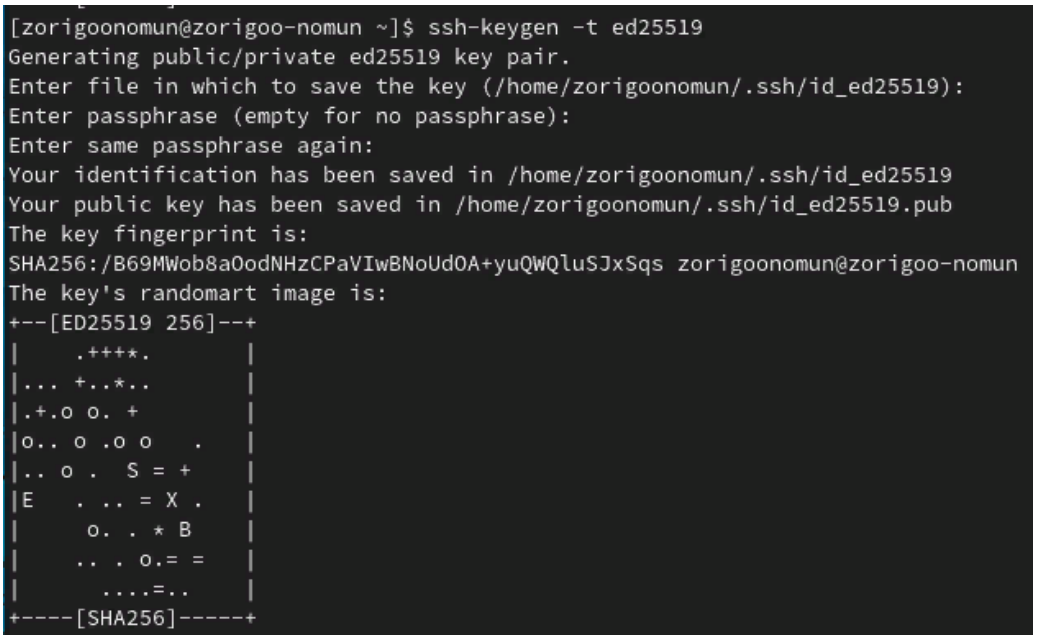

Рис. 3.7: Генерация ssh ключа по алгоритму ed25519

3.4 Создание ключа GPG

Генерирую ключ GPG, затем выбираю тип ключа RSA and RSA, задаю максимальную длину ключа: 4096, оставляю неограниченный срок действия ключа. Далее отвечаю на вопросы программы о личной информации (рис. 3.8).

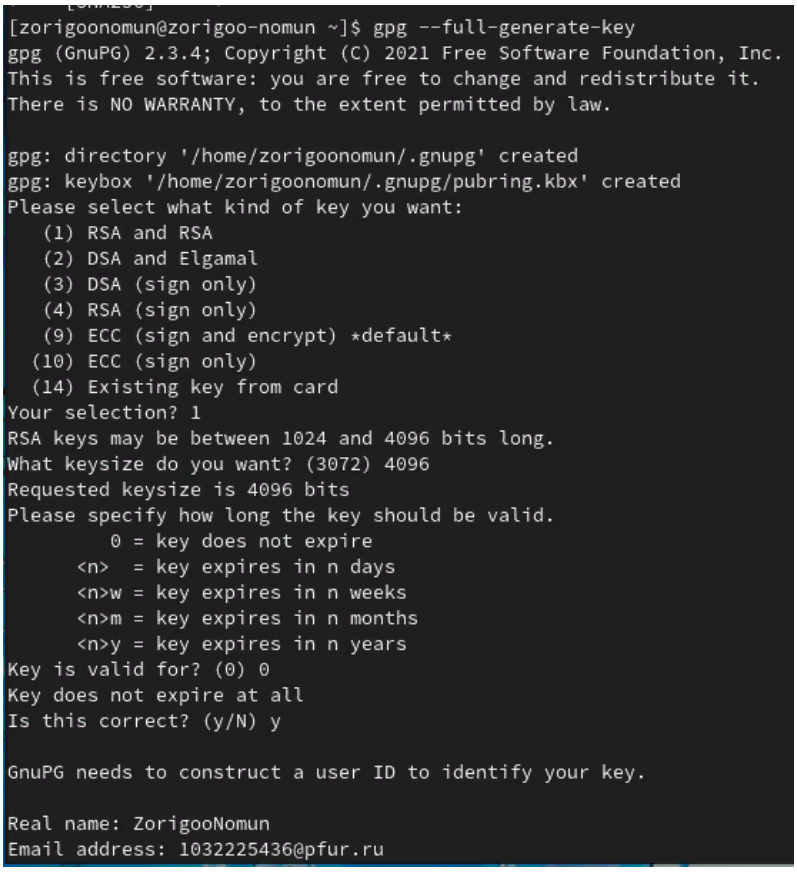

Рис. 3.8: Генерация ключа

Ввожу фразу-пароль для защиты нового ключа (рис. 3.9).
(Забыла сфотографировать)

Рис. 3.9: Защита ключа GPG

3.5 Регистрация на Github

У меня уже был создан аккаунт на Github, соответственно, основные данные аккаунта я так же заполняла и проводила его настройку, поэтому просто вхожу в свой аккаунт (рис. 3.10).

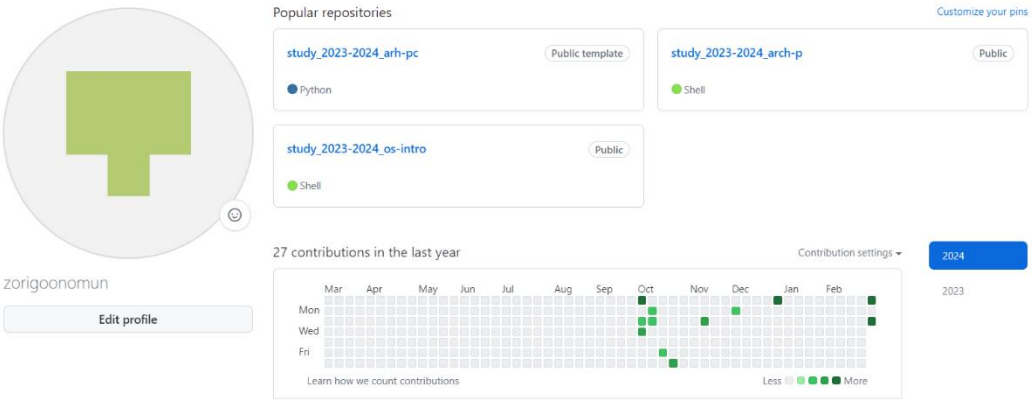

Рис. 3.10: Аккаунт на Github

3.6 Добавление ключа GPG в Github

Вывожу список созданных ключей в терминал, ищу в результате запроса отпечаток ключа (последовательность байтов для идентификации более длинного, по сравнению с самим отпечатком, ключа), он стоит после знака слеша, копирую его в буфер обмена (рис. 3.11).

```
g[zorigoonomun@zorigoo-nomun ~]$ gpg --list-secret-keys --keyid-format LONG
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
/home/zorigoonomun/.gnupg/pubring.kbx
------------------------------------
sec   rsa4096/74E8754254F8F831 2024-02-25 [SC]
      EE17B720674A8F197E4E05A074E8754254F8F831
uid                 [ultimate] zorigoonomun <1032225436@pfur.ru>
ssb   rsa4096/5AD3588E20A97566 2024-02-25 [E]
```

Рис. 3.11: Вывод списка ключей

Ввожу в терминале команду, с помощью которой копирую сам ключ GPG в

буфер обмена, за это отвечает утилита xclip (рис. 3.12).

```
[zorigoonomun@zorigoo-nomun ~]$ gpg --armor --export 74E8754254F8F831 | xclip -s
el clip
```

Рис. 3.12: Копирование ключа в буфер обмена

Открываю настройки GirHub, ищу среди них добавление GPG ключа (рис. 3.13).

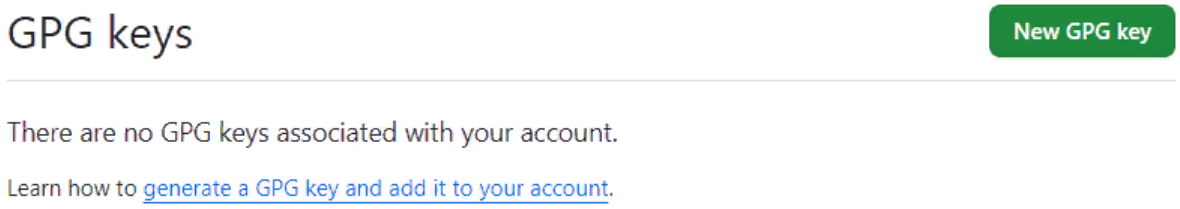

Рис. 3.13: Настройки GitHub

Нажимаю на “New GPG key” и вставляю в поле ключ из буфера обмена (рис. 3.14).

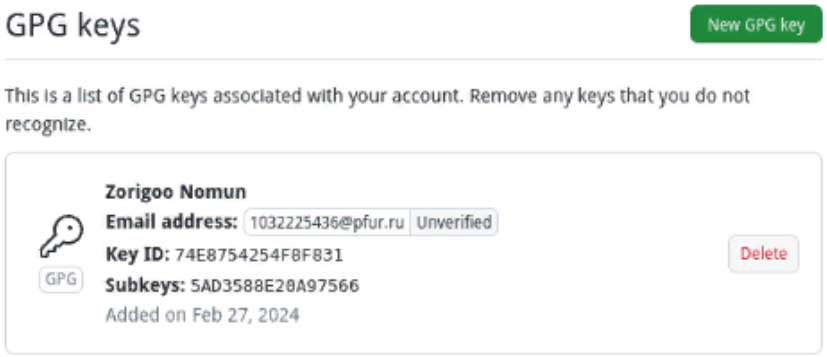

Рис. 3.14: Добавление нового PGP ключа

Я добавила ключ GPG на GitHub (рис. 3.15).

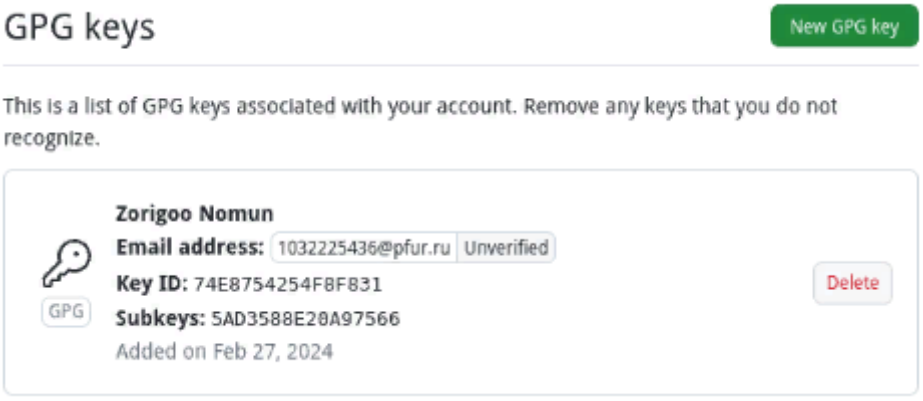

Рис. 3.15: Добавленный ключ GPG

3.7 Настроить подписи Git

Настраиваю автоматические подписи коммитов git: используя введенный ранее email, указываю git использовать его при создании подписей коммитов (рис. 3.16).

```
[zorigoonomun@zorigoo-nomun ~]$ git config --global user.signingkey 74E8754254F8F831
[zorigoonomun@zorigoo-nomun ~]$ git config --global commit.gpgsign true
[zorigoonomun@zorigoo-nomun ~]$ git config --global gpg.program $(which gpg2)
```

Рис. 3.16: Настройка подписей Git

3.8 Настройка gh

Начинаю авторизацию в gh, отвечаю на наводящие вопросы от утилиты, в конце выбираю авторизоваться через браузер (рис. 3.17).

```
[zorigoonomun@zorigoo-nomun ~]$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser
```

Рис. 3.17: Авторизация в gh

Завершаю авторизацию на сайте (рис. 3.18).

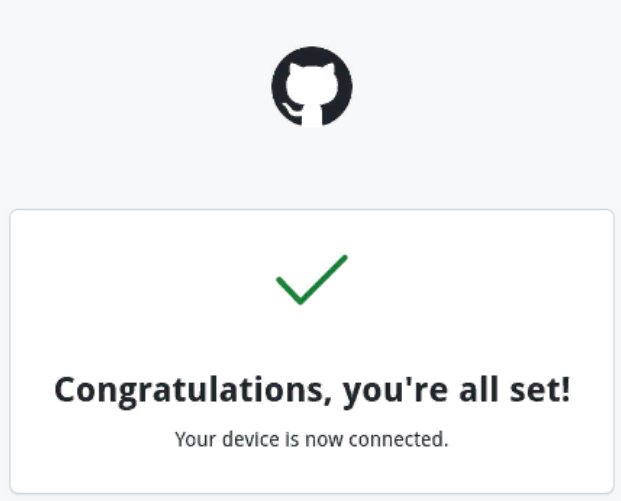

Рис. 3.18: Завершение авторизации через браузер

Вижу сообщение о завершении авторизации под именем evdvorkina (рис. 3.19).

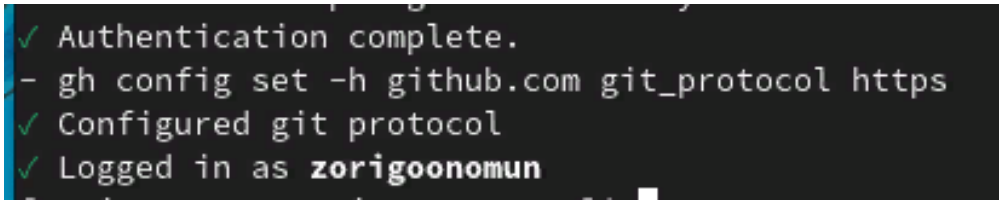

Рис. 3.19: Завершение авторизации

3.9 Создание репозитория курса на основе шаблона

Сначала создаю директорию с помощью утилиты mkdir и флага -p, который позволяет установить каталоги на всем указанном пути. После этого с помощью утилиты cd перехожу в только что созданную директорию “Операционные систе- мы”. Далее в терминале ввожу команду gh repo create study_2022-2023_os-intro –template yamadharma/course-directory-student-trmplate –public, чтобы создать репозиторий на основе шаблона репозитория. После этого клонирую репозиторий к себе в директорию, я указываю ссылку с протоколом https, а не ssh, потому что при авторизации в gh выбрала протокол https (рис. 3.20).

```
[zorigoonomun@zorigoo-nomun ~]$ cd 'Operating systems'
[zorigoonomun@zorigoo-nomun Operating systems]$ git clone --recursive https://github.com/zorigoonomun/st
udy_2023-2024_os-intro.git os-intro
Cloning into '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
Receiving objects: 100% (32/32), 18.60 KiB | 396.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.
Submodule 'template/presentation' (https://github.com/yamadharma/academic-presentation-markdown-template
.git) registered for path 'template/presentation'
```

Рис. 3.20: Создание репозитория

Перехожу в каталог курса с помощью утилиты cd, проверяю содержание каталога с помощью утилиты ls (рис. 3.21).

```
[zorigoonomun@zorigoo-nomun Operating systems]$ cd os-intro
[zorigoonomun@zorigoo-nomun os-intro]$ ls
CHANGELOG.md  COURSE   Makefile      README.en.md        README.md
config        LICENSE  package.json  README.git-flow.md  template
```

Рис. 3.21: Перемещение между директориями

Удаляю лишние файлы с помощью утилиты rm, далее создаю необходимые каталоги используя makefile (рис. 3.22).

```
[zorigoonomun@zorigoo-nomun os-intro]$ rm package.json
[zorigoonomun@zorigoo-nomun os-intro]$ echo os-intro > COURSE
[zorigoonomun@zorigoo-nomun os-intro]$ make
```

Рис. 3.22: Удаление файлов и создание каталогов

Добавляю все новые файлы для отправки на сервер (сохраняю добавленные изменения) с помощью команды git add и комментирую их с помощью git commit (рис. 3.23).

```
[zorigoonomun@zorigoo-nomun os-intro]$ git add .
[zorigoonomun@zorigoo-nomun os-intro]$ git commit -am 'feat(main): make course structure'
[master 7d57d97] feat(main): make course structure
 2 files changed, 1 insertion(+), 14 deletions(-)
 delete mode 100644 package.json
```

Рис. 3.23: Отправка файлов на сервер

Отправляю файлы на сервер с помощью git push (рис. 3.24).

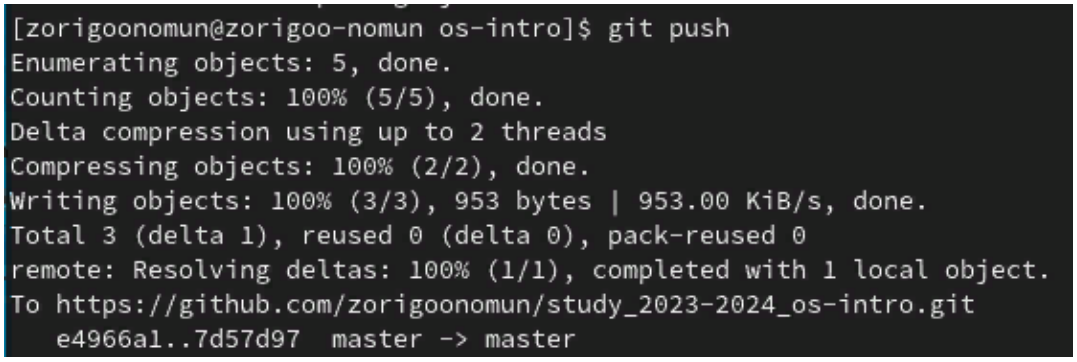

Рис. 3.24: Отправка файлов на сервер

4 Выводы

При выполнении данной лабораторной работы я изучила идеологию и применение средств контроля версий, освоила умение по работе с git.

5 Ответы на контрольные вопросы.

1. Системы контроля версий (VCS) - программное обеспечение для облегчения работы с изменяющейся информацией. Они позволяют хранить несколько версий изменяющейся информации, одного и того же документа, может предоставить доступ к более ранним версиям документа. Используется для работы нескольких человек над проектом, позволяет посмотреть, кто и когда внес какое-либо изменение и т. д. VCS ррименяются для: Хранения понлой истории изменений, сохранения причин всех изменений, поиска причин изменений и совершивших изменение, совместной работы над проектами.

2. Хранилище – репозиторий, хранилище версий, в нем хранятся все документы, включая историю их изменения и прочей служебной информацией. commit – отслеживание изменений, сохраняет разницу в изменениях. История – хранит все изменения в проекте и позволяет при необходимости вернуться/обратиться к нужным данным. Рабочая копия – копия проекта, основанная на версии из хранилища, чаще всего последней версии.

3. Централизованные VCS (например: CVS, TFS, AccuRev) – одно основное хранилище всего проекта. Каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет, затем добавляет изменения обратно в хранилище. Децентрализованные VCS (например: Git, Bazaar) – у каждого пользователя свой вариант репозитория (возможно несколько вариантов), есть возможность добавлять и забирать изменения из любого

репозитория. В отличие от классических, в распределенных (децентрализованных) системах контроля версий центральный репозиторий не является обязательным.

4. Сначала создается и подключается удаленный репозиторий, затем по мере изменения проекта эти изменения отправляются на сервер.

5. Участник проекта перед началом работы получает нужную ему версию проекта в хранилище, с помощью определенных команд, после внесения изменений пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются. К ним можно вернуться в любой момент.

6. Хранение информации о всех изменениях в вашем коде, обеспечение удобства командной работы над кодом.

7. 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

8. git push -all отправляем из локального репозитория все сохраненные изменения в центральный репозиторий, предварительно создав локальный репозиторий и сделав предварительную конфигурацию.

9. Ветвление - один из параллельных участков в одном хранилище, исходящих из одной версии, обычно есть главная ветка. Между ветками, т. е. их концами возможно их слияние. Используются для разработки новых функций.

10. Во время работы над проектом могут создаваться файлы, которые не следуют добавлять в репозиторий. Например, временные файлы. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл .gitignore с помощью сервисов.

Список литературы

1. Лабораторная работа № 2 [Электронный ресурс] URL: https://esystem.rudn.ru/mod/page/vie