

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

дисциплина: Операционные системы

Студент: Алмазова Елизавета

Группа: НПМбд-01-21

МОСКВА

2022 г.

Цель работы: изучить идеологию и применение средств контроля версий, освоить умения по работе с git.

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.

Теоретическое введение:

Системы контроля версий – это программное обеспечение для облегчения работы над изменяющейся информацией, которое часто применяют при работе нескольких человек над одним проектом. Они позволяют фиксировать изменения в файлах и кто их внес, отслеживать и разрешать возникающие конфликты: производить откат, совмещать изменения, сделанные разными участниками, заблокировать файлы для изменения и вручную выбрать версию файлов. Также они могут иметь и дополнительные функции, например, поддержка работы с несколькими версиями путем сохранения общей истории изменений до точки ветвления и собственные истории каждой ветви.

Обычно основное дерево проекта хранится в локальном или удаленном репозитории, к которому настроен доступ для участников проекта. В классических системах используется централизованная модель, т.е. файлы хранятся в едином репозитории. Большинство функций по управлению версиями выполняется специальным сервером. Пользователь перед началом работы с помощью команд получает нужную версию файлов. После внесения изменений он размещает новую версию в хранилище. При этом предыдущие версии удалять не будут, к ним можно вернуться в любой момент. Сервер также может сохранять не полную версию измененных файлов, а производить дельта-компрессию, т.е. сохранять только изменения между последовательными версиями, уменьшая объем хранимых данных. В распределенных системах центральный репозиторий не является обязательным.

Задачи, решаемые VCS:

- Обеспечение командной и распределенной работы над проектом без специального инструментария, предоставление возможности изменять файлы, не мешая работе других пользователей;
- Сохранение исходного кода, хранение всех предыдущих версий или, для уменьшения объема, изменений между последовательными версиями;
- Ведение журнала изменений: кто из участников, когда и какие изменения внес, причем доступ к нему можно ограничить;
- Отслеживание и разрешение конфликтов, возникающих при работе нескольких человек над одним проектом;

- Автоматическое или ручное объединение (совмещение) изменений;
- Ручной выбор нужной версии;
- Откат изменений;
- Настраиваемая блокировка файлов для изменения, предоставление привилегированного доступа только одному пользователю, работающему с файлом;
- Поддержка работы с несколькими версиями одного файла, сохранение общей истории изменений до точки ветвления версий и собственные изменения каждой ветви.

Действия с VCS при единоличной работе с хранилищем:

- Создаем локальный репозиторий.
- Сначала делаем предварительную конфигурацию, указывая имя и email с помощью команд `git config --global user.name "<Name Surname>"`, `git config --global user.email "<email>"` соответственно.
- Настраиваем utf-8 в выводе сообщений git: `git config --global core.quotePath false`.
- С помощью команды `cd` переходим в папку, где хотим создать репозиторий, с помощью `mkdir` создаем каталог, переходим в него с помощью команды `cd` и командой `git init` создаем основное дерево репозитория – появляется каталог `.git` для хранения истории изменений.

Файлы добавляются в локальный репозиторий так:

- `git add [имя файла]`
- `git commit -am '<Комментарий>'`

`Git status` покажет изменения в рабочем каталоге с последней ревизии.

Иногда создаются файлы, которые не требуется добавлять в репозиторий. Тогда можно прописать шаблоны игнорируемых файлов в файл `.gitignore` в корневом каталоге репозитория. Шаблоны можно прописать с помощью сервисов, для этого нужно получить список имеющихся шаблонов с помощью `curl -L -s https://www.gitignore.io/api/list`, затем скачать шаблон, например, на C++ `curl -L -s https://www.gitignore.io/api/c++ >> .gitignore`.

Порядок работы с общим хранилищем VCS:

Проверка и получение изменений из центрального репозитория (до начала процедуры в локальное дерево не должно было вноситься никаких изменений):

- `git checkout master`
- `git pull`
- `git checkout -b [имя ветки]`

После завершения изменений нужно проверить, какие файлы изменились с помощью `git statud`. При необходимости удаляем лишние файлы, которые не нужно отправлять в репозиторий. Просматриваем текст изменений на предмет соответствия правилам ведения чистым коммитов с помощью `git diff`. Если какие-то файлы не должны попасть в коммит, помечаем только те,

изменения которых нужно сохранить, с помощью `git add` и `git rm` с нужными опциями. Если нужно сохранить все изменения, используем “`git add .`”. Сохраняем изменения, поясняя, что было сделано через `git commit –am ‘<Комментарий>’`. Отправляем в репозиторий через `git push origin <имя ветки>` или `git push`.

Основные команды `git`:

- Создание основного дерева репозитория – `git init`;
- Получение обновлений текущего дерева из центрального репозитория – `git pull`;
- Отправка произведенных изменений локального дерева в центральный репозиторий – `git push`;
- Просмотр списка измененных файлов в текущей директории – `git status`;
- Просмотр текущих изменений – `git diff`;
- Сохранение текущих изменений – `git add .` (все изменения)/`git add [имена файлов]` (некоторые изменения);
- Удаление из индекса репозитория – `git rm [имена файлов]`;
- Сохранение добавленных изменений – `git commit –am ‘Комментарий’/git commit` (внесение комментария будет через встроенный редактор);
- Создание новой ветки, базирующейся на текущей – `git checkout –b [имя ветки]`;
- Переключение на некоторую ветку – `git checkout [имя ветки]`;
- 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 – `git branch –d [имя ветки]` (удаление локальной ветки, уже слитой с основным деревом)/`git branch –D [имя ветки]` (принудительное удаление локальной ветки)/ `git push origin: [имя ветки]` (удаление ветки с центрального репозитория);
- Проверка, на какой ветке находится пользователь – `git branch`;
- Завершение работы на ветке – `git flow <ветка> finish`;
- Инициализация структуры `git-flow` в репозитории – `git flow init`;
- Создание функциональной ветки – `git flow feature start feature_branch`;
- Окончание работы с функциональной веткой – `git flow feature finish feature_branch`;
- Создание ветки выпуска – `git flow release start 1.0.0`;
- Окончание работы с веткой выпуска – `git flow release finish 1.0.0`;
- Создание ветки исправления – `git flow hotfix start hotfix_branch`;
- Окончание работы с веткой исправления – `git flow hotfix finish hotfix_branch`.

Ход работы:

1. Я создала и заполнила учетную запись на сайте <https://github.com>. Логин – это мой логин в дисплейном классе, eaalmazova (рис.1).

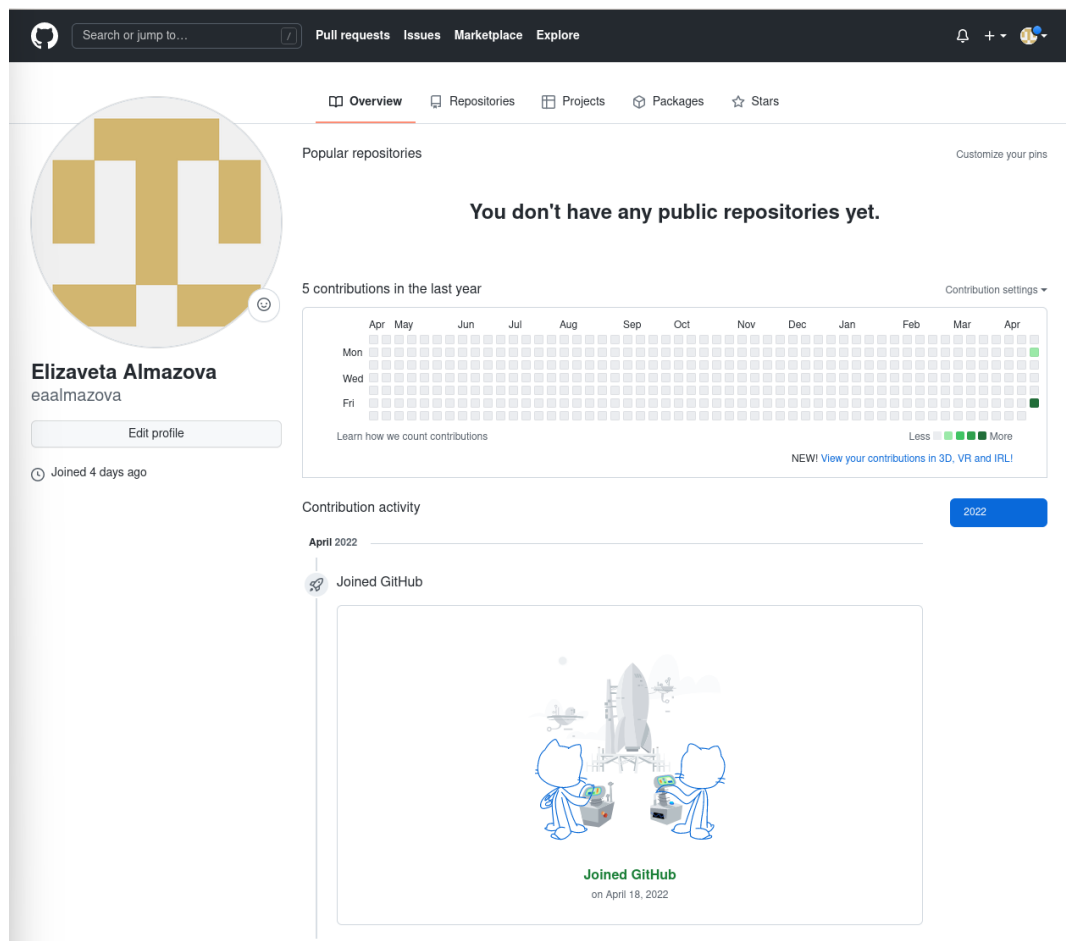


Рисунок 1 – Моя учетная запись на github.com.

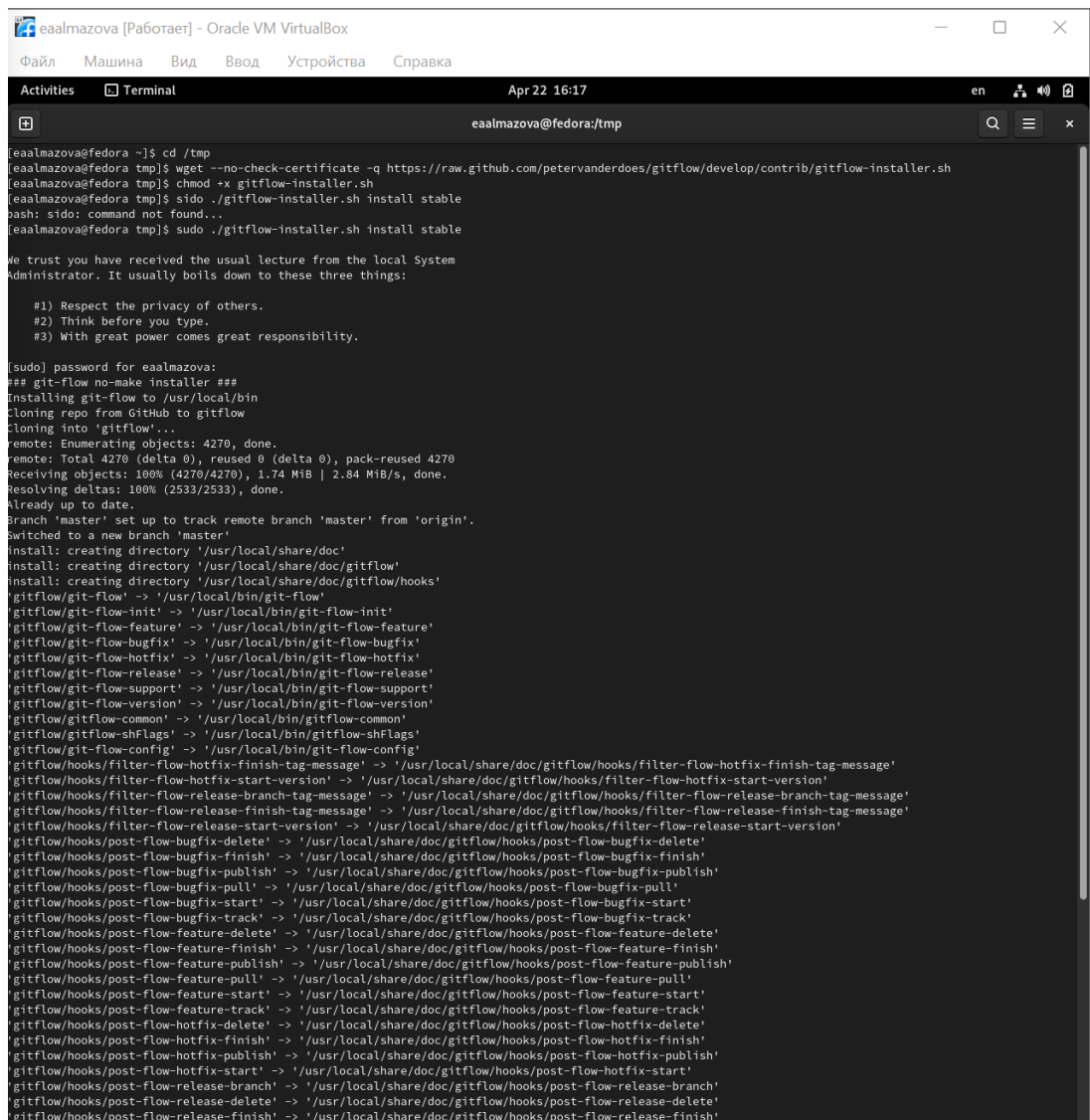
2. Я установила git-flow вручную, поскольку оно удалено из репозитория, с помощью следующего набора команд (рис.2, 3, 4):

- `cd /tmp`
- `wget --no-check-certificate -q https://raw.githubusercontent.com/petervanderdoes/gitflow/develop/contrib/gitflow-installer.sh`
- `chmod +x gitflow-installer.sh`
- `sudo ./gitflow-installer.sh install stable.`

3. Затем я установила gh с помощью команды `sudo dnf install gh` (рис. 3).

4. Следующим шагом стала базовая настройка git (рис.4):

- задание имени и email владельца репозитория (`git config --global user.name "Name Surname"`, `git config --global user.email "work@mail"`);
- настройка utf-8 в выводе сообщений git (`git config --global core.quotepath false`);
- задание имени master начальной ветки (`git config --global init.defaultBranch master`);
- настройка параметров autocrlf (`git config --global core.autocrlf input`) и safecrlf (`git config --global core.safecrlf warn`).



```
eaalmazova [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка

Activities  Terminal  Apr 22 16:17  en  [audio icon]  [share icon]

eaalmazova@fedora:tmp

[eaalmazova@fedora ~]$ cd /tmp
[eaalmazova@fedora tmp]$ wget --no-check-certificate -q https://raw.githubusercontent.com/petervanderdoes/gitflow/develop/contrib/gitflow-installer.sh
[eaalmazova@fedora tmp]$ chmod +x gitflow-installer.sh
[eaalmazova@fedora tmp]$ sudo ./gitflow-installer.sh install stable
bash: sudo: command not found...
[eaalmazova@fedora tmp]$ sudo ./gitflow-installer.sh install stable

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for eaalmazova:
## git-flow no-make installer ##
Installing git-flow to /usr/local/bin
Cloning repo from GitHub to gitflow
Cloning into 'gitflow'...
remote: Enumerating objects: 4270, done.
remote: Total 4270 (delta 0), reused 0 (delta 0), pack-reused 4270
Receiving objects: 100% (4270/4270), 1.74 MiB | 2.84 MiB/s, done.
Resolving deltas: 100% (2533/2533), done.
Already up to date.
Branch 'master' set up to track remote branch 'master' from 'origin'.
Switched to a new branch 'master'
Install: creating directory '/usr/local/share/doc'
Install: creating directory '/usr/local/share/doc/gitflow'
Install: creating directory '/usr/local/share/doc/gitflow/hooks'
'gitflow/git-flow' -> '/usr/local/bin/git-flow'
'gitflow/git-flow-init' -> '/usr/local/bin/git-flow-init'
'gitflow/git-flow-feature' -> '/usr/local/bin/git-flow-feature'
'gitflow/git-flow-bugfix' -> '/usr/local/bin/git-flow-bugfix'
'gitflow/git-flow-hotfix' -> '/usr/local/bin/git-flow-hotfix'
'gitflow/git-flow-release' -> '/usr/local/bin/git-flow-release'
'gitflow/git-flow-support' -> '/usr/local/bin/git-flow-support'
'gitflow/git-flow-version' -> '/usr/local/bin/git-flow-version'
'gitflow/git-flow-common' -> '/usr/local/bin/git-flow-common'
'gitflow/gitflow-shFlags' -> '/usr/local/bin/gitflow-shFlags'
'gitflow/git-flow-config' -> '/usr/local/bin/git-flow-config'
'gitflow/hooks/filter-flow-hotfix-finish-tag-message' -> '/usr/local/share/doc/gitflow/hooks/filter-flow-hotfix-finish-tag-message'
'gitflow/hooks/filter-flow-hotfix-start-version' -> '/usr/local/share/doc/gitflow/hooks/filter-flow-hotfix-start-version'
'gitflow/hooks/filter-flow-release-branch-tag-message' -> '/usr/local/share/doc/gitflow/hooks/filter-flow-release-branch-tag-message'
'gitflow/hooks/filter-flow-release-finish-tag-message' -> '/usr/local/share/doc/gitflow/hooks/filter-flow-release-finish-tag-message'
'gitflow/hooks/filter-flow-release-start-version' -> '/usr/local/share/doc/gitflow/hooks/filter-flow-release-start-version'
'gitflow/hooks/post-flow-bugfix-delete' -> '/usr/local/share/doc/gitflow/hooks/post-flow-bugfix-delete'
'gitflow/hooks/post-flow-bugfix-finish' -> '/usr/local/share/doc/gitflow/hooks/post-flow-bugfix-finish'
'gitflow/hooks/post-flow-bugfix-publish' -> '/usr/local/share/doc/gitflow/hooks/post-flow-bugfix-publish'
'gitflow/hooks/post-flow-bugfix-pull' -> '/usr/local/share/doc/gitflow/hooks/post-flow-bugfix-pull'
'gitflow/hooks/post-flow-bugfix-start' -> '/usr/local/share/doc/gitflow/hooks/post-flow-bugfix-start'
'gitflow/hooks/post-flow-bugfix-track' -> '/usr/local/share/doc/gitflow/hooks/post-flow-bugfix-track'
'gitflow/hooks/post-flow-feature-delete' -> '/usr/local/share/doc/gitflow/hooks/post-flow-feature-delete'
'gitflow/hooks/post-flow-feature-finish' -> '/usr/local/share/doc/gitflow/hooks/post-flow-feature-finish'
'gitflow/hooks/post-flow-feature-publish' -> '/usr/local/share/doc/gitflow/hooks/post-flow-feature-publish'
'gitflow/hooks/post-flow-feature-pull' -> '/usr/local/share/doc/gitflow/hooks/post-flow-feature-pull'
'gitflow/hooks/post-flow-feature-start' -> '/usr/local/share/doc/gitflow/hooks/post-flow-feature-start'
'gitflow/hooks/post-flow-feature-track' -> '/usr/local/share/doc/gitflow/hooks/post-flow-feature-track'
'gitflow/hooks/post-flow-hotfix-delete' -> '/usr/local/share/doc/gitflow/hooks/post-flow-hotfix-delete'
'gitflow/hooks/post-flow-hotfix-finish' -> '/usr/local/share/doc/gitflow/hooks/post-flow-hotfix-finish'
'gitflow/hooks/post-flow-hotfix-publish' -> '/usr/local/share/doc/gitflow/hooks/post-flow-hotfix-publish'
'gitflow/hooks/post-flow-hotfix-start' -> '/usr/local/share/doc/gitflow/hooks/post-flow-hotfix-start'
'gitflow/hooks/post-flow-release-branch' -> '/usr/local/share/doc/gitflow/hooks/post-flow-release-branch'
'gitflow/hooks/post-flow-release-delete' -> '/usr/local/share/doc/gitflow/hooks/post-flow-release-delete'
'gitflow/hooks/post-flow-release-finish' -> '/usr/local/share/doc/gitflow/hooks/post-flow-release-finish'
```

Рисунок 2 – Установка git-flow (1 часть).



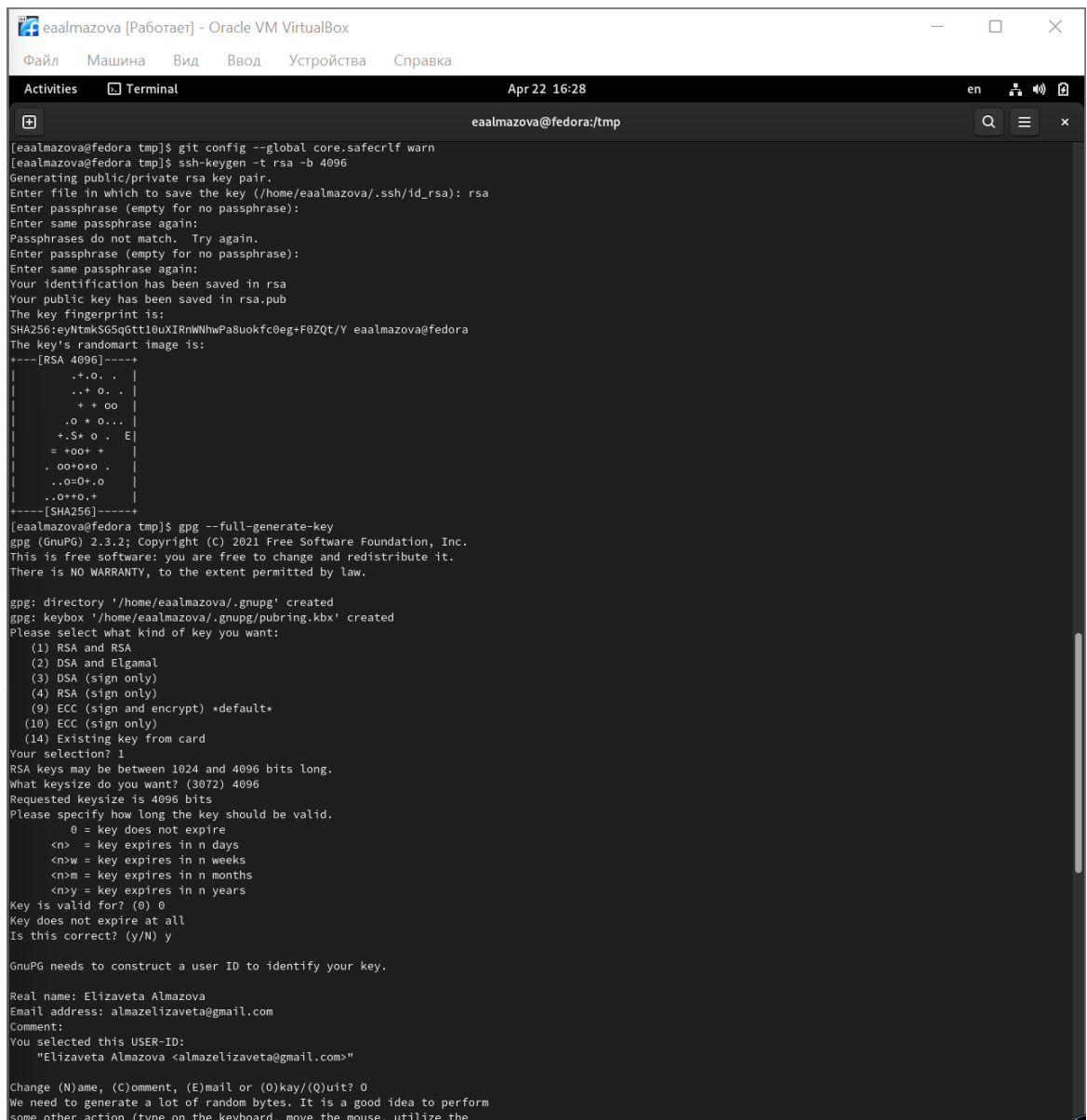
```
gitflow/hooks/post-flow-release-publish' -> '/usr/local/share/doc/gitflow/hooks/post-flow-release-publish'
gitflow/hooks/post-flow-release-start' -> '/usr/local/share/doc/gitflow/hooks/post-flow-release-start'
gitflow/hooks/post-flow-release-track' -> '/usr/local/share/doc/gitflow/hooks/post-flow-release-track'
gitflow/hooks/pre-flow-feature-delete' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-delete'
gitflow/hooks/pre-flow-feature-finish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-finish'
gitflow/hooks/pre-flow-feature-publish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-publish'
gitflow/hooks/pre-flow-feature-pull' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-pull'
gitflow/hooks/pre-flow-feature-start' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-start'
gitflow/hooks/pre-flow-feature-track' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-track'
gitflow/hooks/pre-flow-hotfix-delete' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-hotfix-delete'
gitflow/hooks/pre-flow-hotfix-finish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-hotfix-finish'
gitflow/hooks/pre-flow-hotfix-publish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-hotfix-publish'
gitflow/hooks/pre-flow-hotfix-start' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-hotfix-start'
gitflow/hooks/pre-flow-release-branch' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-branch'
gitflow/hooks/pre-flow-release-delete' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-delete'
gitflow/hooks/pre-flow-release-finish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-finish'
gitflow/hooks/pre-flow-release-publish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-publish'
gitflow/hooks/pre-flow-release-start' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-start'
gitflow/hooks/pre-flow-release-track' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-track'
eaalmazova@fedora tmp]$
```

Рисунок 3 – Установка git-flow (2 часть).

```
gitflow/hooks/post-flow-release-start' -> '/usr/local/share/doc/gitflow/hooks/post-flow-release-start'
gitflow/hooks/post-flow-release-delete' -> '/usr/local/share/doc/gitflow/hooks/post-flow-release-delete'
gitflow/hooks/pre-flow-feature-delete' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-delete'
gitflow/hooks/pre-flow-feature-finish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-finish'
gitflow/hooks/pre-flow-feature-publish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-publish'
gitflow/hooks/pre-flow-feature-pull' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-pull'
gitflow/hooks/pre-flow-feature-start' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-start'
gitflow/hooks/pre-flow-feature-track' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-feature-track'
gitflow/hooks/pre-flow-hotfix-delete' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-hotfix-delete'
gitflow/hooks/pre-flow-hotfix-finish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-hotfix-finish'
gitflow/hooks/pre-flow-hotfix-publish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-hotfix-publish'
gitflow/hooks/pre-flow-hotfix-start' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-hotfix-start'
gitflow/hooks/pre-flow-release-branch' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-branch'
gitflow/hooks/pre-flow-release-delete' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-delete'
gitflow/hooks/pre-flow-release-finish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-finish'
gitflow/hooks/pre-flow-release-publish' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-publish'
gitflow/hooks/pre-flow-release-start' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-start'
gitflow/hooks/pre-flow-release-track' -> '/usr/local/share/doc/gitflow/hooks/pre-flow-release-track'
[eaalmazova@fedora tmp]$ sudo dnf install gh
Fedora 35 - x86_64
Fedora 35 openh264 (From Cisco) - x86_64
Fedora Modular 35 - x86_64
Fedora 35 - x86_64 - Updates
Fedora Modular 35 - x86_64 - Updates
Last metadata expiration check: 0:00:01 ago on Fri 22 Apr 2022 04:20:48 PM MSK.
Dependencies resolved.
=====
Package Architecture Version Repository Size
=====
Installing:
gh x86_64 2.7.0-1.fc35 updates 6.8 M
Transaction Summary
=====
Install 1 Package
Total download size: 6.8 M
Installed size: 32 M
Is this ok [y/N]: y
Downloading Packages:
gh-2.7.0-1.fc35.x86_64.rpm 7.9 MB/s | 6.8 MB 00:00
Total 6.8 MB/s | 6.8 MB 00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installing : gh-2.7.0-1.fc35.x86_64 1/1
Running scriptlet: gh-2.7.0-1.fc35.x86_64 1/1
Verifying : gh-2.7.0-1.fc35.x86_64 1/1
Installed:
gh-2.7.0-1.fc35.x86_64
Complete!
[eaalmazova@fedora tmp]$ git config --global user.name eaalmazova
[eaalmazova@fedora tmp]$ git config --global user.email almazelizaveta@gmail.com
[eaalmazova@fedora tmp]$ git config --global core.quotepath false
[eaalmazova@fedora tmp]$ git config --global init.defaultBranch master
[eaalmazova@fedora tmp]$ git config --global core.autocrlf input
[eaalmazova@fedora tmp]$ git config --global core.safecrlf warn
[eaalmazova@fedora tmp]$
```

Рисунок 4 – Установка git-flow (3 часть), установка gh и настройка git.

5. Я создала ключ ssh по алгоритму rsa с ключем размером 4096 бит (команда: `ssh-keygen -t rsa -b 4096` и ключ pgr (команда генерации: `gpg --full-generate-key`): тип RSA and RSA, размер 4096, срок действия не истекает никогда. Для сохранения в ключе я также ввела личную информацию: имя и адрес электронной почты, указанный на GitHub (рис.5,6).



```
eaalmazova [Работаer] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка

Activities  Terminal  Apr 22 16:28  en  [icons]

eaalmazova@fedora:tmp

[eaalmazova@fedora tmp]$ git config --global core.safecrlf warn
[eaalmazova@fedora tmp]$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/eaalmazova/.ssh/id_rsa): rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Passphrases do not match. Try again.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in rsa
Your public key has been saved in rsa.pub
The key fingerprint is:
SHA256:eyNtmkSG5qGt10uXIRnWWhwPa8uokfc0eg+F0ZQt/Y eaalmazova@fedora
The key's randomart image is:
+----[RSA 4096]-----+
|.+.o. .|
|.+.o. .|
|+.+oo|
|.o*o...|
|+.S*o .E|
|=+oo+ +|
|.oo+o+o .|
|..o=0+.o|
|.o++o.+|
+----[SHA256]-----+
[eaalmazova@fedora tmp]$ gpg --full-generate-key
gpg (GnuPG) 2.3.2; 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.

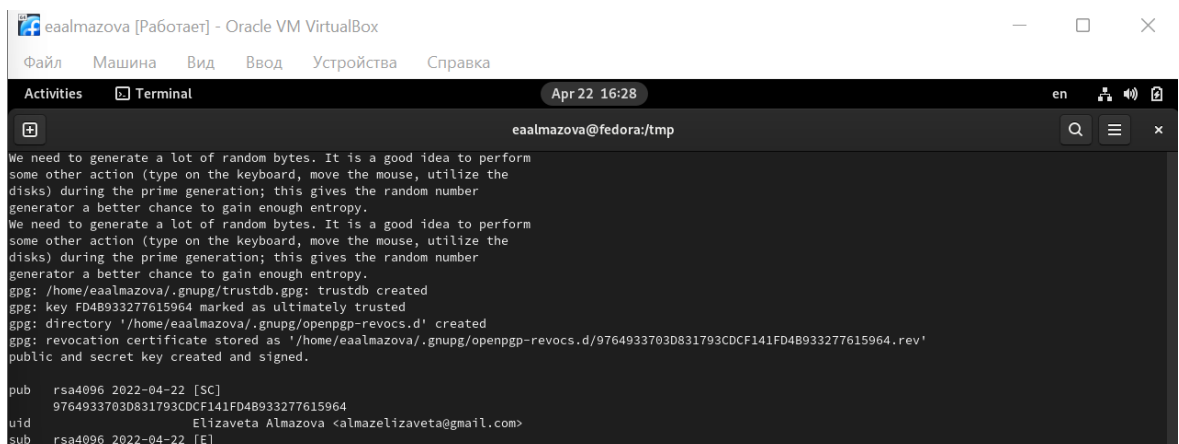
gpg: directory '/home/eaalmazova/.gnupg' created
gpg: keybox '/home/eaalmazova/.gnupg/pubring.kbx' created
Please select what kind of key you want:
(1) RSA and RSA
(2) DSA and Elgamal
(3) DSA (sign only)
(4) RSA (sign only)
(9) ECC (sign and encrypt) *default*
(10) ECC (sign only)
(14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096
Requested keysize is 4096 bits
Please specify how long the key should be valid.
0 = key does not expire
<n> = key expires in n days
<n>w = key expires in n weeks
<n>m = key expires in n months
<n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Elizaveta Almazova
Email address: almazelizaveta@gmail.com
Comment:
You selected this USER-ID:
"Elizaveta Almazova <almazelizaveta@gmail.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? 0
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
```

Рисунок 5 – Создание ключей ssh и gpg.



```
eaalmazova [Работаer] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка

Activities  Terminal  Apr 22 16:28  en  [icons]

eaalmazova@fedora:tmp

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /home/eaalmazova/.gnupg/trustdb.gpg: trustdb created
gpg: key FD4B933277615964 marked as ultimately trusted
gpg: directory '/home/eaalmazova/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/eaalmazova/.gnupg/openpgp-revocs.d/9764933703D831793C0CF141FD4B933277615964.rev'
public and secret key created and signed.

pub   rsa4096 2022-04-22 [SC]
      9764933703D831793C0CF141FD4B933277615964
uid           Elizaveta Almazova <almazelizaveta@gmail.com>
sub   rsa4096 2022-04-22 [E]
```

Рисунок 6 – Продолжение создания ключа gpg.

6. Я вывела список ключей с помощью команды `gpg --list-secret-keys --keyid-format LONG`. Для копирования ключа в буфер обмена я также установила

утилиту xclip с помощью команды `sudo dnf install xclip` (рис.7). Затем я скопировала сгенерированный PGP ключ в буфер обмена (команда: `gpg --armor --export FD4B933277615964 | xclip -sel clip`, где FD4B933277615964 – отпечаток ключа) и добавила его как новый GPG ключ в настройках GitHub (рис.8).

```
[eaalmazova@fedora tmp]$ 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/eaalmazova/.gnupg/pubring.kbx
-----
sec   rsa4096/FD4B933277615964 2022-04-22 [SC]
      9764933703D831793CDCF141F04B933277615964
uid   [ultimate] Elizaveta Almazova <almazelizaveta@gmail.com>
ssb   rsa4096/304C12B2B07613E8 2022-04-22 [E]

[eaalmazova@fedora tmp]$ sudo dnf install xclip
[sudo] password for eaalmazova:
Last metadata expiration check: 0:05:33 ago on Fri 22 Apr 2022 04:20:48 PM MSK.
Dependencies resolved.
=====
Package                        Architecture      Version           Repository        Size
=====
Installing:
xclip                          x86_64            0.13-15.git11cba61.fc35  fedora            36 k
=====
Transaction Summary
=====
Install 1 Package

Total download size: 36 k
Installed size: 62 k
Is this ok [y/N]: y
Downloading Packages:
xclip-0.13-15.git11cba61.fc35.x86_64.rpm                                413 kB/s | 36 kB  00:00
-----
Total                                                                    58 kB/s | 36 kB  00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing      :
  Installing     : xclip-0.13-15.git11cba61.fc35.x86_64                1/1
  Running scriptlet: xclip-0.13-15.git11cba61.fc35.x86_64                1/1
  Verifying      : xclip-0.13-15.git11cba61.fc35.x86_64                1/1

Installed:
xclip-0.13-15.git11cba61.fc35.x86_64

Complete!
[eaalmazova@fedora tmp]$ gpg --armor --export FD4B933277615964 | xclip -sel clip
[eaalmazova@fedora tmp]$
```

Рисунок 7 – Вывод списка ключей, установка xclip и копирование ключа.

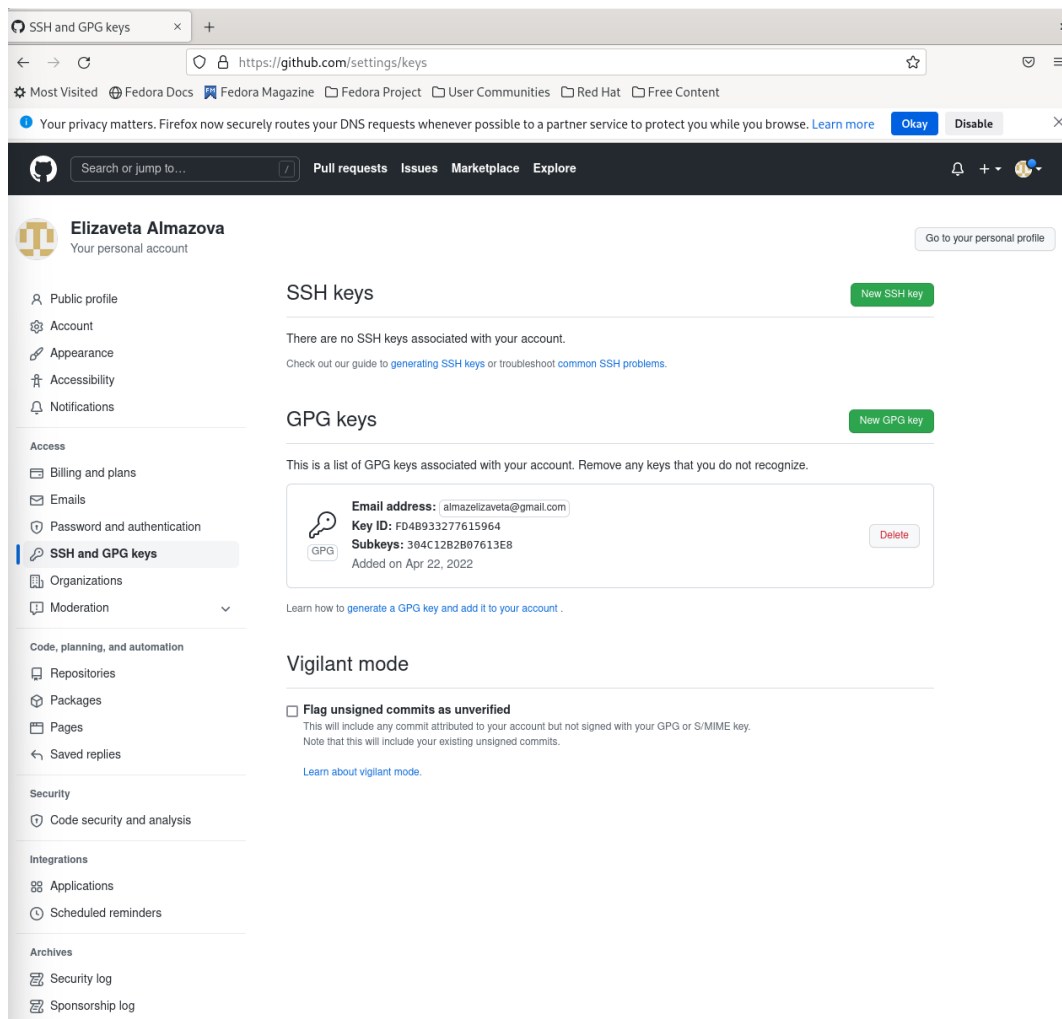


Рисунок 8 – Добавленный GPG ключ на сайте github.com

7. Я настроила автоматические подписи коммитов git с помощью следующих команд (рис.9):

- git config --global user.signingkey FD4B933277615964
- git config --global commit.gpgsign true
- git config --global gpg.program \$(which gpg2)

```
[eaalmazova@fedora tmp]$ git config --global user.signingkey FD4B933277615964
[eaalmazova@fedora tmp]$ git config --global commit.gpgsign true
[eaalmazova@fedora tmp]$ git config --global gpg.program $(which gpg2)
```

Рисунок 9 – Настройка автоматических подписей git.

8. Я авторизовалась с помощью команды gh auth login, ответив на несколько вопросов (рис.10, 11).

```
[eaalmazova@fedora tmp]$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? SSH
? Generate a new SSH key to add to your GitHub account? No
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: B180-8127
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 eaalmazova
```

Рисунок 10 – Авторизация в терминале.

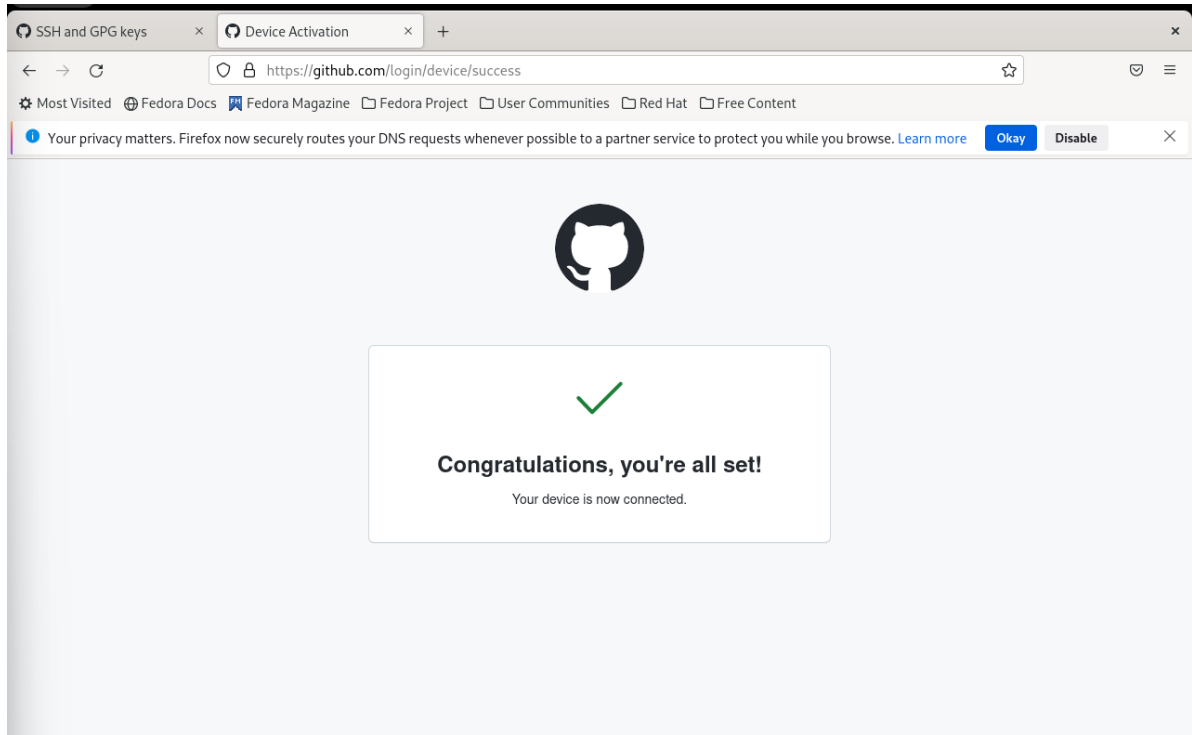


Рисунок 11 – Уведомление об успешной авторизации.

9. Я создала репозиторий курса на основе шаблона с помощью следующих команд (рис.12):

- `mkdir -p ~/work/study/2021-2022/"Операционные системы"`
- `cd ~/work/study/2021-2022/"Операционные системы"`
- `gh repo create study_2021-2022_os-intro --template=yamadharma/course-directory-student-template --public`
- `git clone --recursive https://github.com/eaalmazova/study_2021-2022_os-intro.git os-intro`

```

[eaalmazova@fedora tmp]$ mkdir -p ~/work/study/2021-2022/"Операционные системы"
[eaalmazova@fedora tmp]$ cd ~/work/study/2021-2022/"Операционные системы"
[eaalmazova@fedora Операционные системы]$ gh repo create study_2021-2022_os-intro --template=yamadharma/course-directory-student-template --public
Created repository eaalmazova/study_2021-2022_os-intro on GitHub
[eaalmazova@fedora Операционные системы]$ git clone --recursive git@github.com:eaalmazova/study_2021-2022_os-intro.git os-intro
Cloning into 'os-intro'...
The authenticity of host 'github.com (140.82.121.4)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvV6TuJJhpZisF/zLDA0zPMSvHdkr4UvCQOu.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
[eaalmazova@fedora Операционные системы]$ git clone --recursive https://github.com/eaalmazova/study_2021-2022_os-intro.git os-intro
Cloning into 'os-intro'...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 20 (delta 2), reused 15 (delta 2), pack-reused 0
Receiving objects: 100% (20/20), 12.50 KiB | 12.50 MiB/s, done.
Resolving deltas: 100% (2/2), done.
Submodule 'template/presentation' (https://github.com/yamadharma/academic-presentation-markdown-template.git) registered for path 'template/presentation'
Submodule 'template/report' (https://github.com/yamadharma/academic-laboratory-report-template.git) registered for path 'template/report'
Cloning into '/home/eaalmazova/work/study/2021-2022/Операционные системы/os-intro/template/presentation'...
remote: Enumerating objects: 42, done.
remote: Counting objects: 100% (42/42), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 42 (delta 9), reused 40 (delta 7), pack-reused 0
Receiving objects: 100% (42/42), 31.19 KiB | 709.00 KiB/s, done.
Resolving deltas: 100% (9/9), done.
Cloning into '/home/eaalmazova/work/study/2021-2022/Операционные системы/os-intro/template/report'...
remote: Enumerating objects: 78, done.
remote: Counting objects: 100% (78/78), done.
remote: Compressing objects: 100% (52/52), done.
remote: Total 78 (delta 31), reused 69 (delta 22), pack-reused 0
Receiving objects: 100% (78/78), 292.27 KiB | 1.59 MiB/s, done.
Resolving deltas: 100% (31/31), done.
Submodule path 'template/presentation': checked out '3eae7b7586f8a9aded2b506cd1018e625b228b93'
Submodule path 'template/report': checked out 'df7b2ef80f8def3b9a496f8695277469a1a7842a'

```

Рисунок 12 – Создание репозитория курса на основе шаблона.

10. Я перешла в каталог курса с помощью команды `cd ~/work/study/2021-2022/"Операционные системы"/os-intro`. Там я удалила лишний файл с помощью команды `rm package.json` и создала необходимый каталог командой `make COURSE=os-intro`. Затем я отправила файлы на сервер с помощью следующих команд (рис. 13,14,15):

- `git add .`
- `git commit -am 'feat(main): make course structure'`
- `git push`

```

ealalmazova@fedora Операционные системы$ cd ~/work/study/2021-2022/"Операционные системы"/os-intro
ealalmazova@fedora os-intro$ rm package.json
ealalmazova@fedora os-intro$ make COURSE=os-intro
ealalmazova@fedora os-intro$ git add .
ealalmazova@fedora os-intro$ git commit -am 'feat(main): make course structure'
[master 1149e4d] feat(main): make course structure
149 files changed, 16590 insertions(+), 14 deletions(-)
create mode 108644 labs/lab01/presentation/Makefile
create mode 108644 labs/lab01/presentation/presentation.md
create mode 108644 labs/lab01/report/Makefile
create mode 108644 labs/lab01/report/bib/cite.bib
create mode 108644 labs/lab01/report/image/placeimg_800_600_tech.jpg
create mode 108644 labs/lab01/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 labs/lab01/report/report.md
create mode 108644 labs/lab02/presentation/Makefile
create mode 108644 labs/lab02/presentation/presentation.md
create mode 108644 labs/lab02/report/Makefile
create mode 108644 labs/lab02/report/bib/cite.bib
create mode 108644 labs/lab02/report/image/placeimg_800_600_tech.jpg
create mode 108644 labs/lab02/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 labs/lab02/report/report.md
create mode 108644 labs/lab03/presentation/Makefile
create mode 108644 labs/lab03/presentation/presentation.md
create mode 108644 labs/lab03/report/Makefile
create mode 108644 labs/lab03/report/bib/cite.bib
create mode 108644 labs/lab03/report/image/placeimg_800_600_tech.jpg
create mode 108644 labs/lab03/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 labs/lab03/report/report.md
create mode 108644 labs/lab04/presentation/Makefile
create mode 108644 labs/lab04/presentation/presentation.md
create mode 108644 labs/lab04/report/Makefile
create mode 108644 labs/lab04/report/bib/cite.bib
create mode 108644 labs/lab04/report/image/placeimg_800_600_tech.jpg
create mode 108644 labs/lab04/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 labs/lab04/report/report.md
create mode 108644 labs/lab05/presentation/Makefile
create mode 108644 labs/lab05/presentation/presentation.md
create mode 108644 labs/lab05/report/Makefile
create mode 108644 labs/lab05/report/bib/cite.bib
create mode 108644 labs/lab05/report/image/placeimg_800_600_tech.jpg
create mode 108644 labs/lab05/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 labs/lab05/report/report.md
create mode 108644 labs/lab06/presentation/Makefile
create mode 108644 labs/lab06/presentation/presentation.md
create mode 108644 labs/lab06/report/Makefile
create mode 108644 labs/lab06/report/bib/cite.bib
create mode 108644 labs/lab06/report/image/placeimg_800_600_tech.jpg
create mode 108644 labs/lab06/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 labs/lab06/report/report.md
create mode 108644 labs/lab07/presentation/Makefile
create mode 108644 labs/lab07/presentation/presentation.md
create mode 108644 labs/lab07/report/Makefile
create mode 108644 labs/lab07/report/bib/cite.bib
create mode 108644 labs/lab07/report/image/placeimg_800_600_tech.jpg
create mode 108644 labs/lab07/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 labs/lab07/report/report.md
create mode 108644 labs/lab08/presentation/Makefile
create mode 108644 labs/lab08/presentation/presentation.md
create mode 108644 labs/lab08/report/Makefile
create mode 108644 labs/lab08/report/bib/cite.bib
create mode 108644 labs/lab08/report/image/placeimg_800_600_tech.jpg
create mode 108644 labs/lab08/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 labs/lab08/report/report.md

```

Рисунок 13 – Управление каталогом курса и отправка файлов на сервер

```

delete mode 108644 package.json
create mode 108644 project-personal/stage1/presentation/Makefile
create mode 108644 project-personal/stage1/presentation/presentation.md
create mode 108644 project-personal/stage1/report/Makefile
create mode 108644 project-personal/stage1/report/bib/cite.bib
create mode 108644 project-personal/stage1/report/image/placeimg_800_600_tech.jpg
create mode 108644 project-personal/stage1/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 project-personal/stage1/report/report.md
create mode 108644 project-personal/stage2/presentation/Makefile
create mode 108644 project-personal/stage2/presentation/presentation.md
create mode 108644 project-personal/stage2/report/Makefile
create mode 108644 project-personal/stage2/report/bib/cite.bib
create mode 108644 project-personal/stage2/report/image/placeimg_800_600_tech.jpg
create mode 108644 project-personal/stage2/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 project-personal/stage2/report/report.md
create mode 108644 project-personal/stage3/presentation/Makefile
create mode 108644 project-personal/stage3/presentation/presentation.md
create mode 108644 project-personal/stage3/report/Makefile
create mode 108644 project-personal/stage3/report/bib/cite.bib
create mode 108644 project-personal/stage3/report/image/placeimg_800_600_tech.jpg
create mode 108644 project-personal/stage3/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 project-personal/stage3/report/report.md
create mode 108644 project-personal/stage4/presentation/Makefile
create mode 108644 project-personal/stage4/presentation/presentation.md
create mode 108644 project-personal/stage4/report/Makefile
create mode 108644 project-personal/stage4/report/bib/cite.bib
create mode 108644 project-personal/stage4/report/image/placeimg_800_600_tech.jpg
create mode 108644 project-personal/stage4/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 project-personal/stage4/report/report.md
create mode 108644 project-personal/stage5/presentation/Makefile
create mode 108644 project-personal/stage5/presentation/presentation.md
create mode 108644 project-personal/stage5/report/Makefile
create mode 108644 project-personal/stage5/report/bib/cite.bib
create mode 108644 project-personal/stage5/report/image/placeimg_800_600_tech.jpg
create mode 108644 project-personal/stage5/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 project-personal/stage5/report/report.md
create mode 108644 project-personal/stage6/presentation/Makefile
create mode 108644 project-personal/stage6/presentation/presentation.md
create mode 108644 project-personal/stage6/report/Makefile
create mode 108644 project-personal/stage6/report/bib/cite.bib
create mode 108644 project-personal/stage6/report/image/placeimg_800_600_tech.jpg
create mode 108644 project-personal/stage6/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 108644 project-personal/stage6/report/report.md
ealalmazova@fedora os-intro$ git push
Username for 'https://github.com': ealalmazova
Password for 'https://ealalmazova@github.com':
remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
fatal: Authentication failed for 'https://github.com:ealalmazova/study_2021-2022_os-intro.git/'
ealalmazova@fedora os-intro$ git push
Username for 'https://github.com': ealalmazova
Password for 'https://ealalmazova@github.com':
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Compressing objects: 100% (16/16), done.
Writing objects: 100% (19/19), 266.53 KiB | 38.87 MiB/s, done.
Total 19 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com:ealalmazova/study_2021-2022_os-intro.git
  1e5ed2f..1149e4d master -> master
ealalmazova@fedora os-intro$

```

Рисунок 14 – Продолжение отправки файлов на сервер.

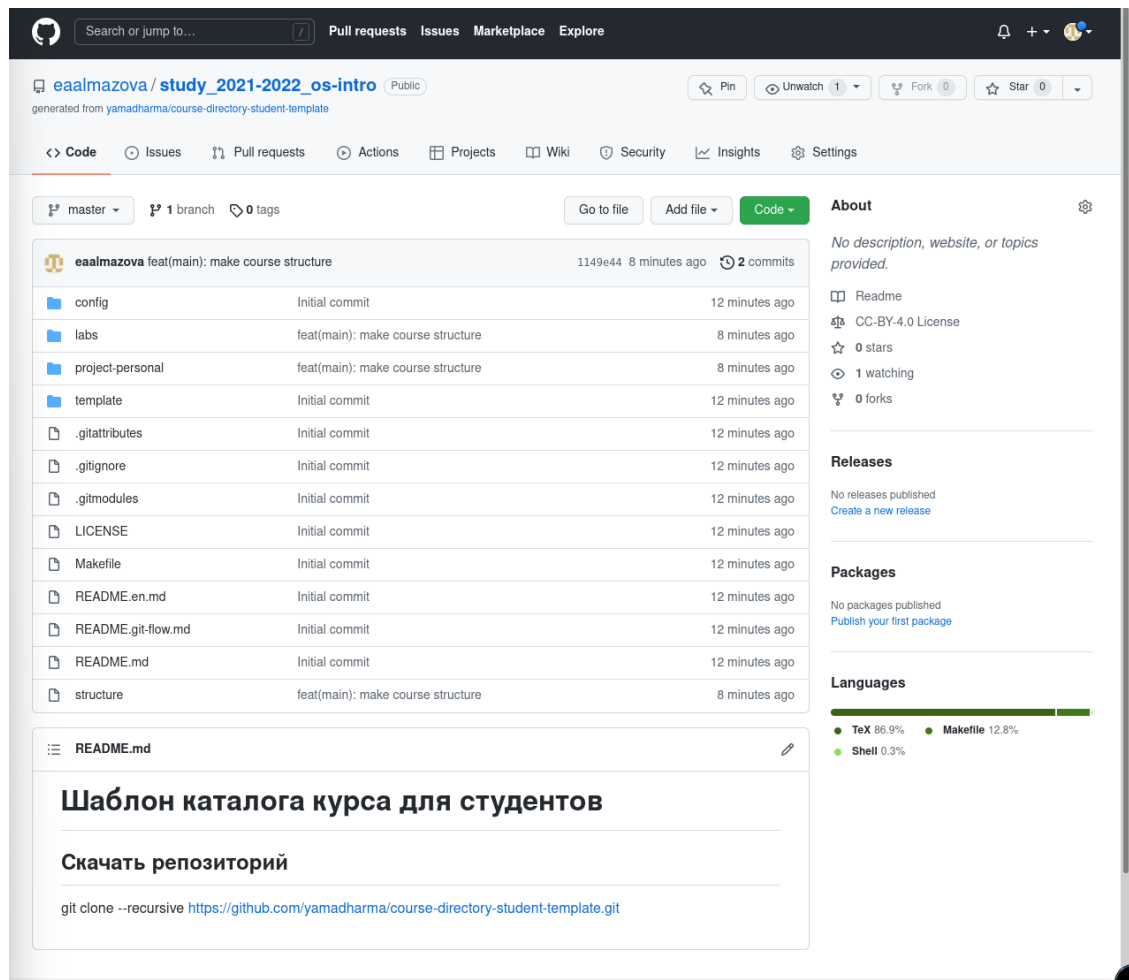


Рисунок 15 – Результат работы с репозиторием курса.

Вывод: в ходе выполнения данной лабораторной работы я изучила идеологию и применение средств контроля версий, освоила умения по работе с git.

Контрольные вопросы:

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначены?

Системы контроля версий – это программное обеспечение для облегчения работы над изменяющейся информацией, которое часто применяют при работе нескольких человек над одним проектом. Они позволяют фиксировать изменения в файлах и кто их внес, отслеживать и разрешать возникающие конфликты: производить откат, совмещать изменения, сделанные разными участниками, заблокировать файлы для изменения и вручную выбрать версию файлов. Также они могут иметь и дополнительные функции, например, поддержка работы с несколькими версиями путем сохранения общей истории изменений до точки ветвления и собственные истории каждой ветви.

Обычно основное дерево проекта хранится в локальном или удаленном репозитории, к которому настроен доступ для участников проекта. В классических системах используется централизованная модель, т.е.

файлы хранятся в едином репозитории. Большинство функций по управлению версиями выполняется специальным сервером. Пользователь перед началом работы с помощью команд получает нужную версию файлов. После внесения изменений он размещает новую версию в хранилище. При этом предыдущие версии удалять не будут, к ним можно вернуться в любой момент. Сервер также может сохранять не полную версию измененных файлов, а производить дельта-компрессию, т.е. сохранять только изменения между последовательными версиями, уменьшая объем хранимых данных. В распределенных системах центральный репозиторий не является обязательным.

Задачи, решаемые VCS:

- Обеспечение командной и распределенной работы над проектом без специального инструментария, предоставление возможности изменять файлы, не мешая работе других пользователей;
- Сохранение исходного кода, хранение всех предыдущих версий или, для уменьшения объема, изменений между последовательными версиями;
- Ведение журнала изменений: кто из участников, когда и какие изменения внес, причем доступ к нему можно ограничить;
- Отслеживание и разрешение конфликтов, возникающих при работе нескольких человек над одним проектом;
- Автоматическое или ручное объединение (совмещение) изменений;
- Ручной выбор нужной версии;
- Откат изменений;
- Настраиваемая блокировка файлов для изменения, предоставление привилегированного доступа только одному пользователю, работающему с файлом;
- Поддержка работы с несколькими версиями одного файла, сохранение общей истории изменений до точки ветвления версий и собственные изменения каждой ветви.

2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.

- Хранилище, также называется репозиторий – это место, где хранятся все файлы вместе с историей их изменения и другой служебной информацией.
- Commit – внесение в репозиторий добавленных изменений, возможно с добавлением комментария.
- История – это исходный код проекта, записи времени, автора и сущности всех внесенных изменений.
- Рабочая копия – копия проекта, срез репозитория в какой-то момент времени и внесенные пользователем в него изменения, хранящиеся на устройстве этого пользователя.

Все файлы проекта хранятся в хранилище. Пользователь работает на

своем устройстве над копией файлов из хранилища, которую называют рабочей копией, и вносит туда некоторые изменения. После окончания работы пользователь делает `commit`, т.е. отправляет измененные файлы в хранилище. В истории отражаются предыдущая версия и новая версия с внесенными пользователем изменениями, их временем и именем пользователя, который их внес.

3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.

Принципы работы централизованных и децентрализованных (распределенных) VCS похожи, отличаются они в основном синтаксисом используемых в работе команд, однако есть некоторые существенные различия. Централизованные системы предполагают наличие единого репозитория. Большинство функций по управлению версиями выполняется специальным сервером. Пользователь перед началом работы с помощью команд получает нужную версию файлов. После внесения изменений он размещает новую версию в хранилище. При этом предыдущие версии удалять не будут, к ним можно вернуться в любой момент. Сервер также может сохранять не полную версию измененных файлов, а производить дельта-компрессию, т.е. сохранять только изменения между последовательными версиями, уменьшая объем хранимых данных. Другие пользователи также могут видеть внесенные изменения. Примеры: CVS, Subversion.

В распределенных системах центральный репозиторий не является обязательным, т.е. каждый пользователь пользуется полной копией проекта со всеми версиями всех файлов. Изменения вносятся группами (`changeset` вместо `commit`). Примеры: Git, Bazaar, Mercurial.

4. Опишите действия с VCS при единоличной работе с хранилищем.

- Создаем локальный репозиторий.
- Сначала делаем предварительную конфигурацию, указывая имя и email с помощью команд `git config --global user.name "<Name Surname>"`, `git config --global user.email "<email>"` соответственно.
- Настраиваем utf-8 в выводе сообщений git: `git config --global core.quotePath false`.
- С помощью команды `cd` переходим в папку, где хотим создать репозиторий, с помощью `mkdir` создаем каталог, переходим в него с помощью команды `cd` и командой `git init` создаем основное дерево репозитория – появляется каталог `.git` для хранения истории изменений.

Файлы добавляются в локальный репозиторий так:

- `git add [имя файла]`
- `git commit -am '<Комментарий>'`

`Git status` покажет изменения в рабочем каталоге с последней ревизии.

Иногда создаются файлы, которые не требуется добавлять в репозиторий.

Тогда можно прописать шаблоны игнорируемых файлов в файл .gitignore в корневом каталоге репозитория. Шаблоны можно прописать с помощью сервисов, для этого нужно получить список имеющихся шаблонов с помощью `curl -L -s https://www.gitignore.io/api/list`, затем скачать шаблон, например, на C++ `curl -L -s https://www.gitignore.io/api/c++ >> .gitignore`.

5. Опишите порядок работы с общим хранилищем VCS.

Проверка и получение изменений из центрального репозитория (до начала процедуры в локальное дерево не должно было вноситься никаких изменений):

- `git checkout master`
- `git pull`
- `git checkout -b [имя ветки]`

После завершения изменений нужно проверить, какие файлы изменились с помощью `git status`. При необходимости удаляем лишние файлы, которые не нужно отправлять в репозиторий. Просматриваем текст изменений на предмет соответствия правилам ведения чистым коммитов с помощью `git diff`. Если какие-то файлы не должны попасть в коммит, помечаем только те, изменения которых нужно сохранить, с помощью `git add` и `git rm` с нужными опциями. Если нужно сохранить все изменения, используем `"git add ."`. Сохраняем изменения, поясняя, что было сделано через `git commit -am '<Комментарий>'`. Отправляем в репозиторий через `git push origin <имя ветки>` или `git push`.

6. Каковы основные задачи, решаемые инструментальным средством git?

- Создание основного дерева репозитория;
- Получение обновлений текущего дерева из центрального репозитория;
- Отправка произведенных изменений локального дерева в центральный репозиторий;
- Просмотр списка измененных файлов в текущей директории;
- Просмотр текущих изменений;
- Сохранение текущих изменений;
- Удаление из индекса репозитория;
- Сохранение добавленных изменений;
- Создание новой ветки, базирующейся на текущей;
- Переключение на некоторую ветку;
- Отправка изменений конкретной ветки в центральный репозиторий;
- Слияние ветки с текущим деревом;
- Удаление ветки;
- Проверка, на какой ветке находится пользователь;
- Завершение работы на ветке;
- Работа с разными видами веток по модели Gitflow.

7. Назовите и дайте краткую характеристику командам git.

- Создание основного дерева репозитория – `git init`;
- Получение обновлений текущего дерева из центрального репозитория – `git pull`;
- Отправка произведенных изменений локального дерева в центральный репозиторий – `git push`;
- Просмотр списка измененных файлов в текущей директории – `git status`;
- Просмотр текущих изменений – `git diff`;
- Сохранение текущих изменений – `git add .` (все изменения)/`git add [имена файлов]` (некоторые изменения);
- Удаление из индекса репозитория – `git rm [имена файлов]`;
- Сохранение добавленных изменений – `git commit -am 'Комментарий'`/`git commit` (внесение комментария будет через встроенный редактор);
- Создание новой ветки, базирующейся на текущей – `git checkout -b [имя ветки]`;
- Переключение на некоторую ветку – `git checkout [имя ветки]`;
- 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 – `git branch -d [имя ветки]` (удаление локальной ветки, уже слитой с основным деревом)/`git branch -D [имя ветки]` (принудительное удаление локальной ветки)/ `git push origin: [имя ветки]` (удаление ветки с центрального репозитория);
- Проверка, на какой ветке находится пользователь – `git branch`;
- Завершение работы на ветке – `git flow <ветка> finish`;
- Инициализация структуры `git-flow` в репозитории – `git flow init`;
- Создание функциональной ветки – `git flow feature start feature_branch`;
- Окончание работы с функциональной веткой – `git flow feature finish feature_branch`;
- Создание ветки выпуска – `git flow release start 1.0.0`;
- Окончание работы с веткой выпуска – `git flow release finish 1.0.0`;
- Создание ветки исправления – `git flow hotfix start hotfix_branch`;
- Окончание работы с веткой исправления – `git flow hotfix finish hotfix_branch`.

8. Приведите примеры использования при работе с локальным и удалённым репозиториями.

`Git diff` можно использовать при работе с центральным репозиторием, чтобы просмотреть текст изменений на предмет соответствия правилам ведения чистых коммитов. Чтобы при работе с локальным репозиторием посмотреть изменения в рабочем каталоге, которые были сделаны с момента последней ревизии, можно воспользоваться командой `git status`.

9. Что такое и зачем могут быть нужны ветви (branches)?

Ветвь – это направление разработки, независимое от других. Она представляет собой копию части хранилища, в которую можно вносить изменения, не влияющую на другие ветки. В целом они помогают организовать рабочий процесс и исправлять ошибки в рабочей среде.

Модель Gitflow предлагает следующую модель: вместо одной ветки master используются две ветки. В основной ветке master хранится официальная история релиза и каждому коммиту присваивается номер версии, ветка разработки develop предназначена для объединения всех функций. Под каждую новую функцию отводится собственная функциональная ветка feature на основе develop. После завершения работы над функцией ветка сливается с develop. Когда в develop оказывается достаточное количество функций для выпуска, из этой ветки создается ветка выпуска release. С момента создания ветки выпуска добавлять новые функции больше нельзя, допускается только отладка, создание документации и решение других задач. После подготовки релиза release сливается с master и ей присваивается номер версии. После нужно сделать слияние с веткой develop, в которой с момента создания ветки могли возникнуть изменения. Это позволяет дорабатывать текущий выпуск, в то время как другая команда продолжает работу над функциями для следующего релиза. Ветви исправления hotfix используются для быстрого внесения изменений в рабочие релизы, они создаются от ветки master. После исправления их объединяют с release и develop, ветка master должна быть помечена обновленным номером версии. Наличие такой специальной ветки позволяет решать проблемы, не дожидаясь следующего релиза и не прерывая остальную часть рабочего процесса.

10. Как и зачем можно игнорировать некоторые файлы при commit?

Игнорируемые файлы должны быть помечены явным образом. Правила игнорирования обычно задаются в текстовом файле .gitignore в корневом каталоге репозитория, однако можно определить и несколько файлов в разных каталогах репозитория. Внутри содержится шаблон подстановки для сопоставления имен файлов с подстановочными знаками. Шаблоны можно прописать с помощью сервисов. Для этого нужно получить список имеющихся шаблонов с помощью `curl -L -s https://www.gitignore.io/api/list`, затем скачать шаблон, например, на C++ `curl -L -s https://www.gitignore.io/api/c++ >> .gitignore`.

Обычно игнорируются файлы для конкретной платформы, автоматически созданные файлы из систем сборки, артефакты сборки или файлы, которые по какой-то причине не должны попадать в коммиты. Например, кэши зависимостей, скомпилированный код, каталоги для выходных данных сборки, файлы, сгенерированные во время выполнения, скрытые системные файлы, файлы с конфиденциальной информацией и другие.

Это делается, чтобы ненужные файлы не засоряли проект, так как они или являются конфиденциальными, или не являются частью проекта, которой следует делиться, так как могут быть созданы пользователями самостоятельно, или не требуются при разработке проекта.