

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

Отчет

Иван Борисович Салиндер

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	11
4.1	Установка программного обеспечения	11
4.2	Базовая настройка git	11
4.3	Создание ключа SSH	12
4.4	Создание ключа PGP	12
4.5	Регистрация на Github	13
4.6	Добавление PGP на Github	13
4.7	Подписывание коммитов git и настройка подписей git	15
4.8	Авторизация в gh	15
4.9	Создание локального репозитория	16
5	Выводы	18
6	Ответы на контрольные вопросы	19
	Список литературы	22

Список иллюстраций

4.1	Задаю имя и email владельца репозитория	11
4.2	Настраиваем utf-8 в выводе сообщений git	11
4.3	Задаем имя начальной ветки	11
4.4	Задаем параметры autocrlf и safecrlf	12
4.5	Генерация ssh ключа по алгоритму ed25519	12
4.6	Генерация ключа GPG	13
4.7	Мой аккаунт на Github	13
4.8	Вывод списка ключей	14
4.9	Настройки Github	14
4.10	Добавление ключ GPG	14
4.11	Добавленный ключ	15
4.12	Настройка подписей git	15
4.13	Авторизация в gh	15
4.14	Завершение авторизации в gh	16
4.15	Создание репозитория	16
4.16	Копирование файлов репозитория на устройство	16
4.17	Удаление файлов и создание каталогов	17
4.18	Структурирование фрагментов	17
4.19	Отправка файлов на сервер	17

Список таблиц

1 Цель работы

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

2 Задание

Создать базовую конфигурацию для работы с git. Создать ключ SSH Создать ключ PGP Настроить подписи git Зарегистрироваться на Github Создать локальный каталог по выполнению заданий по предмету

3 Теоретическое введение

Системы контроля версий. Общие понятия

Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется.

В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять не полную версию изменённых файлов, а производить так называемую дельта-компрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.

Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить (слить) изменения, сделанные разными

участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависимости от настроек блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, привилегированный доступ только одному пользователю, работающему с файлом.

Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например, они могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Кроме того, обычно доступна информация о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить.

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

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

Примеры использования git

Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды `git` с различными опциями. Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией. Основные команды `git` Перечислим наиболее часто используемые команды `git`. Создание основного дерева репозитория: `git init` Получение обновлений (изменений) текущего дерева из центрального репозитория:

`git pull` Отправка всех произведённых изменений локального дерева в центральный репозиторий:

git push Просмотр списка изменённых файлов в текущей директории: git status
Просмотр текущих изменений:

git diff Сохранение текущих изменений: добавить все изменённые и/или созданные файлы и/или каталоги: git add . добавить конкретные изменённые и/или созданные файлы и/или каталоги: git add имена_файлов удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm имена_файлов Сохранение добавленных изменений: сохранить все добавленные изменения и все изменённые файлы: git commit -am 'Описание коммита' сохранить добавленные изменения с внесением комментария через встроенный редактор: git commit создание новой ветки, базирующейся на текущей: git checkout -b имя_ветки переключение на некоторую ветку: git checkout имя_ветки (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) отправка изменений конкретной ветки в центральный репозиторий:

git push origin имя_ветки слияние ветки с текущим деревом:

git merge --no-ff имя_ветки Удаление ветки: удаление локальной уже слитой с основным деревом ветки: git branch -d имя_ветки принудительное удаление локальной ветки: git branch -D имя_ветки удаление ветки с центрального репозитория: git push origin :имя_ветки

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 удаляем лишние файлы, которые не хотим отправлять в центральный репозиторий.

Затем полезно посмотреть 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 сохраняем изменения, поясняя, что было сделано: `git commit -am "Some commit message"` Отправляем изменения в центральный репозиторий: `git push origin имя_ветки` или `git push`

4 Выполнение лабораторной работы

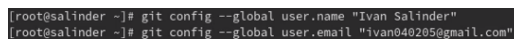
4.1 Установка программного обеспечения

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

Установка git и gh

4.2 Базовая настройка git

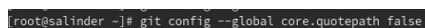
Задаю в качестве имени и email владельца репозитория свое имя, фамилию и электронную почту (рис.).



```
root@salinder ~]# git config --global user.name "Ivan Salinder"
root@salinder ~]# git config --global user.email "ivan040205@gmail.com"
```

Рис. 4.1: Задаю имя и email владельца репозитория

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



```
root@salinder ~]# git config --global core.quotePath false
```

Рис. 4.2: Настраиваем utf-8 в выводе сообщений git

Начальной ветке задаю имя master (рис.)



```
root@salinder ~]# git config --global init.defaultBranch master
```

Рис. 4.3: Задаем имя начальной ветки

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

```
root@salinder ~]# git config --global core.autocrlf input
root@salinder ~]# git config --global core.safecrlf warn
```

Рис. 4.4: Задаем параметры autocrlf и safecrlf

4.3 Создание ключа SSH

Задаю ключ ssh размером 4096 бит по алгоритму rsa (рис)

Генерация ssh ключа по алгоритму rsa

Создаю ключ ssh по алгоритму ed25519 (рис)

```
root@salinder ~]# ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_ed25519
Your public key has been saved in /root/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:xJfITyOGHvf8gf+hfkNrfDnQwzIwGAqlyfGLSG7aj9Q root@salinder
The key's randomart image is:
+--[ED25519 256]--+
|..o+ ..          |
|o+o.o. o o .     |
|+ o + O B        |
|o. . . B O =     |
|. o E . S = + o   |
|. o      o =.+    |
|. .      oo=.o    |
|. .      o++.    |
|. .      .oo.o.   |
+-----[SHA256]-----+
```

Рис. 4.5: Генерация ssh ключа по алгоритму ed25519

4.4 Создание ключа PGP

Генерирую ключ PGP, затем выбираю тип ключа RSA and RSA, задаю максимальную длину ключа 4096, оставляю неограниченный срок действия ключа. (рис.)

```

[root@salinder ~]# 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? 1
длина ключей RSA может быть от 1024 до 4096.
Какой размер ключа Вам необходим? (3072) 4096
Запрошенный размер ключа - 4096 бит
Выберите срок действия ключа.
  0 = не ограничен
  <n> = срок действия ключа - n дней
  <n>w = срок действия ключа - n недель
  <n>m = срок действия ключа - n месяцев
  <n>y = срок действия ключа - n лет
Срок действия ключа? (0)

```

Рис. 4.6: Генерация ключа GPG

4.5 Регистрация на Github

У меня уже был создан на Github, соответственно, настройки аккаунта я также заполнил. Поэтому я просто вхожу в свой аккаунт (рис.)

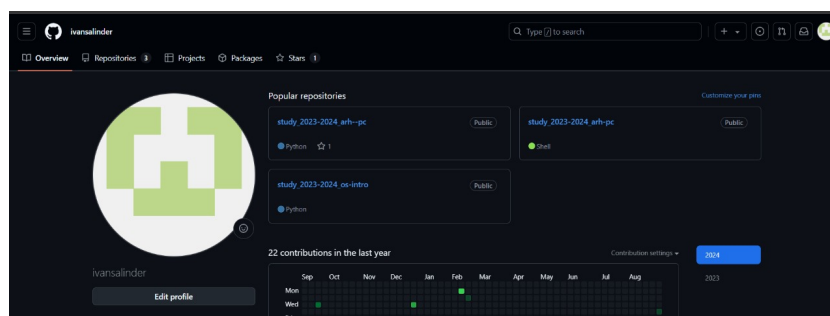


Рис. 4.7: Мой аккаунт на Github

4.6 Добавление PGP на Github

Вывожу список отпечаток ключа в терминал, ищу в результате запроса отпечаток ключа (последовательность байтов для идентификации более длинного ключа), он стоит после знака слева. Копирую его в буфер обмена (рис)

```

[root@salinder ~]# gpg --list-secret-keys --keyid-format LONG
gpg: проверка таблицы доверия
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: глубина: 0 доверенных: 1 подписанных: 0 доверие: 0-, 0q, 0n, 0m, 0f, 1u
/root/.gnupg/pubring.kbx
-----
sec  rsa4096/D4A99F992BAD2932 2024-09-03 [SC]
      92E6B00CBAB83E37ACEACB76D4A99F992BAD2932
uid          [ абсолютно ] IvanIvan <ivan640205@gmail.com>
ssb  rsa4096/B2AFC8B1EDFA97FE 2024-09-03 [E]

```

Рис. 4.8: Вывод списка ключей

Ввожу в терминал команду, с помощью которой копирую сам ключ PGP в буфер обмена, за это отвечает утилита хclip (рис.)

Копирование ключа в буфер обмена

Открываю настройки Github, среди них находим добавление GPG ключа (рис.)

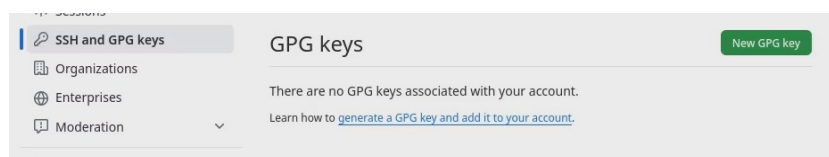


Рис. 4.9: Настройки Github

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

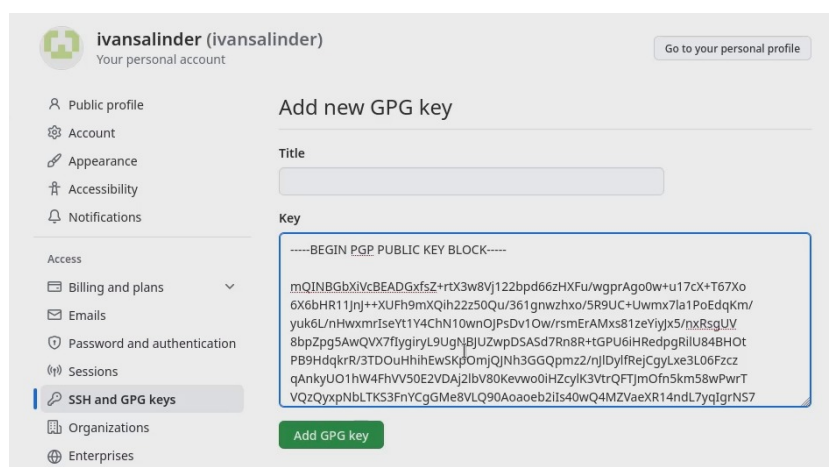


Рис. 4.10: Добавление ключ GPG

Добавляю ключ GPG на Github (рис. [fig.013?]) (рис.)

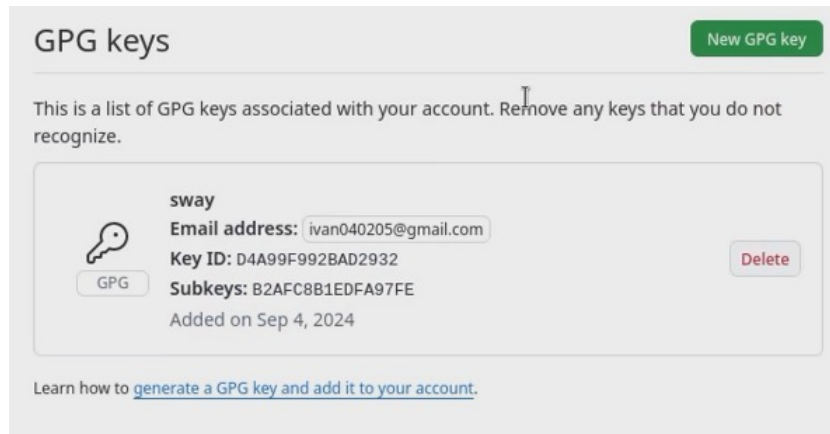


Рис. 4.11: Добавленный ключ

4.7 Подписывание коммитов git и настройка подписей git

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

```
[root@salinder ~]# gpg --armor --export ivan040205@gmail.com | xclip -sel clip
[root@salinder ~]# git config --global user.signingkey ivan040205@gmail.com
[root@salinder ~]# git config --global commit.gpgsign true
[root@salinder ~]# git config --global gpg.program $(which gpg2)
[root@salinder ~]#
```

Рис. 4.12: Настройка подписей git

4.8 Авторизация в gh

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

```
[root@salinder ~]# gh auth login
? 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? [Use arrows to move, type to filter]
```

Рис. 4.13: Авторизация в gh

Завершаю авторизацию на сайте (рис.)



Рис. 4.14: Завершение авторизации в gh

4.9 Создание локального репозитория

Сначала создаю директорию с помощью утилиты `mkdir` и флага `-p`, который позволяет установить каталоги на всем указанном пути. После этого перехожу в созданную директорию и создаю репозиторий с помощью команды `gh repo create` (рис.)

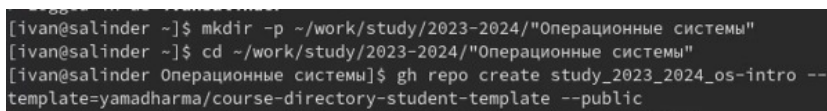


Рис. 4.15: Создание репозитория

После клонирую репозиторий к себе в директорию. Копирую с протоколом `https`, а не `ssh`, потому что при авторизации в `gh` выбрал протокол `https` (рис.)

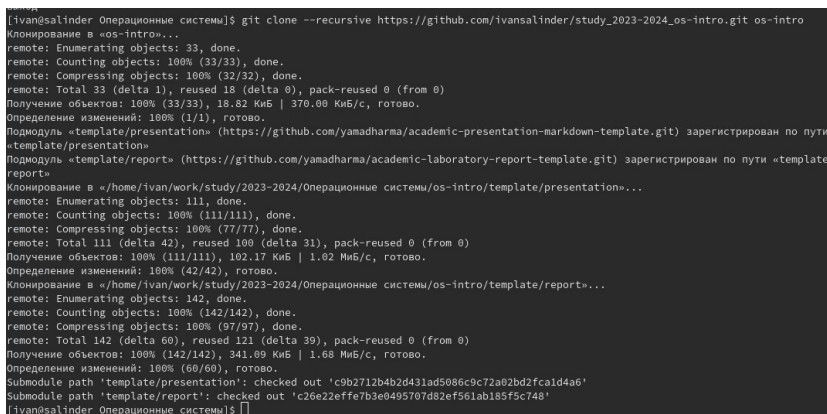


Рис. 4.16: Копирование файлов репозитория на устройство

Удаляю лишние файлы с помощью утилиты `rm` и создаю каталоги (рис.)


```
[ivan@salinder os-intro]$ rm package.json
[ivan@salinder os-intro]$ echo os-intro > COURSE
[ivan@salinder os-intro]$ make
usage:
```

Рис. 4.17: Удаление файлов и создание каталогов

Добавляю все новые файлы для отправки на сервер (рис.)

```
[ivan@salinder os-intro]$ git commit -am 'feat(main): make course structure'
[master 8b81d5a] feat(main): make course structure
485 files changed, 98413 insertions(+), 14 deletions(-)
create mode 108644 labs/README.ru.md
create mode 108644 labs/lab01/presentation/.projectile
create mode 108644 labs/lab01/presentation/.texlabroot
create mode 108644 labs/lab01/presentation/Makefile
create mode 108644 labs/lab01/presentation/image/kulyabov.jpg
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
```

Рис. 4.18: Структурирование фрагментов

Отправляю файлы на сервер (рис.)

```
[ivan@salinder os-intro]$ git push
Перечисление объектов: 40%, готово.
Подсчет объектов: 100% (40/40), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (38/38), готово.
Запись объектов: 100% (38/38), 342.18 КиБ | 21.39 МБ/с, готово.
Total 38 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To https://github.com/ivansalinder/study_2023-2024_os-intro.git
b28bb18..8e88a4e master -> master
[ivan@salinder os-intro]$
```

Рис. 4.19: Отправка файлов на сервер

5 Выводы

При выполнении данной лабораторной работы я изучил идеологию и применение средств контроля версий. Усвоил умение по работе с git

6 Ответы на контрольные вопросы

1. Системы контроля версий - 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 какие-либо изменения и тд. VCS применяются для : хранения полной истории изменений, сохранения причин всех изменений, поиска причин изменений и совершивших изменения. Также для совместной работы над проектами
2. Хранилище - репозиторий, хранилище версий, в нем хранятся все документы, в том числе история изменений и прочая служебная информация. Commit - отслеживание изменений, сохраняет разницу в изменениях. История - хранит все изменения в проекте и при необходимости позволяет вернуться/обратиться к нужным данным. Рабочая копия - копия проекта, основанная на версии из хранилища, обычно самая последняя версия
3. Централизованная VCS (например CVS, TFS, AccuRev) - одно основное хранилище всего проекта. Каждый пользователь копирует себе необходимые файлы из этого репозитория, изменяет, затем добавляет к себе в хранилище. Децентрализованные VCS (Git, Bazaar) - у каждого пользователя свой вариант репозитория, есть возможность добавлять и забирать изменения из любого репозитория. В отличие от классических, в распределенных системах контроля версий центральных 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 и к ним можно обратиться в любой момент
6. Хранение информации о всех изменениях в коде, обеспечение удобства командной работы над кодом
7. Перечислим наиболее часто используемые команды git. Создание основного дерева репозитория: `git init` Получение обновлений (изменений) текущего дерева из центрального репозитория: `git pull` Отправка всех произведённых изменений локального дерева в центральный репозиторий: `git push` Просмотр списка изменённых файлов в текущей директории: `git status` Просмотр текущих изменений: `git diff`

Сохранение текущих изменений: добавить все изменённые и/или созданные файлы и/или каталоги: `git add .` добавить конкретные изменённые и/или созданные файлы и/или каталоги: `git add имена_файлов` удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): `git rm имена_файлов`

Сохранение добавленных изменений: сохранить все добавленные изменения и все изменённые файлы: `git commit -am 'Описание коммита'` сохранить добавленные изменения с внесением комментария через встроенный редактор: `git commit` создание новой ветки, базирующейся на текущей: `git checkout -b имя_ветки` переключение на некоторую ветку: `git checkout имя_ветки` (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) отправка изменений конкретной ветки в центральный репозиторий: `git push origin имя_ветки` слияние ветки с текущим деревом: `git merge --no-ff имя_ветки`

Удаление ветки: удаление локальной уже слитой с основным деревом ветки: `git branch -d имя_ветки` принудительное удаление локальной ветки: `git branch -D имя_ветки` удаление ветки с центрального репозитория: `git push origin :имя_ветки`

8. Git Push - all отправляем из локального репозитория все сохраненные изменения в центральный репозиторий, предварительно создав локальный репозиторий и сделав предварительную информацию

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

10. Во время работы над проектом могут создаваться файлы, которые не следует 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/view>