

Лабораторная работа №2

Первоначальная настройка git

Чемоданова Ангелина Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	12
5	Выводы	19
6	Контрольные вопросы	20

Список иллюстраций

4.1	Регистрация на GitHub	12
4.2	Установка git-flow	13
4.3	Установка gh	13
4.4	Первичная настройка параметров git	14
4.5	Создание ключа ssh по алгоритму rsa с ключем размером 4096 бит	14
4.6	Создание ключа ssh по алгоритму ed25519	15
4.7	Создание ключа gpg	15
4.8	Отпечаток приватного ключа	16
4.9	Добавление pgr-ключ на гитхаб	16
4.10	Настройка автоматических подписей коммитов git	16
4.11	Настройка gh и подтверждение авторизации	17
4.12	Добавление ssh-ключа на гитхаб	17
4.13	Создание репозитория курса	17
4.14	Настройка каталога курса	18
4.15	Настройка каталога курса	18

Список таблиц

1 Цель работы

1. Изучить идеологию и применение средств контроля версий.
2. 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 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. Благодаря тому, что Git является распределённой системой контроля версий, резервн

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

```
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)

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
```

4 Выполнение лабораторной работы

Для начала регистрируемся на гитхабе. (рис. [4.1]).

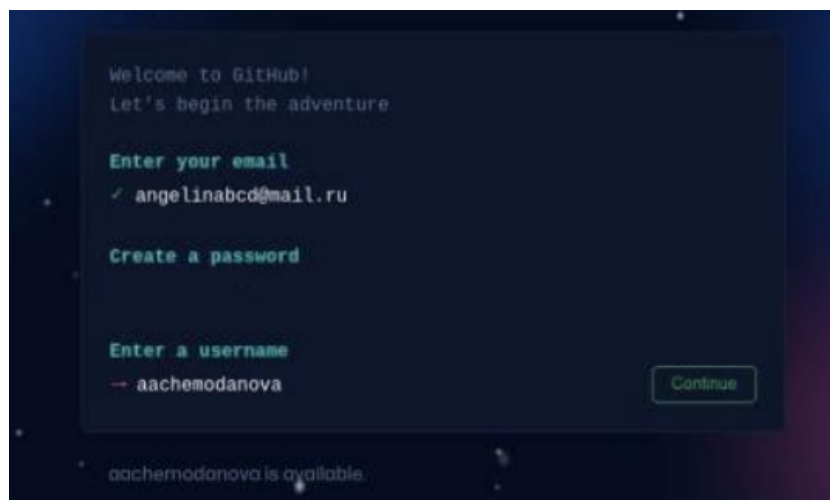


Рис. 4.1: Регистрация на GitHub

Затем необходимо установить git-flow. (рис. [4.2]).

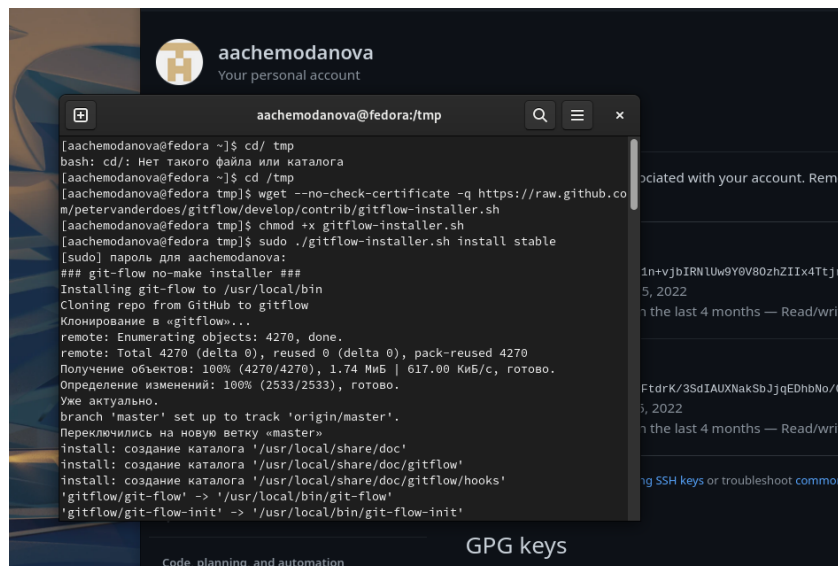


Рис. 4.2: Установка git-flow

Устанавливаем gh. (рис. [4.3]).

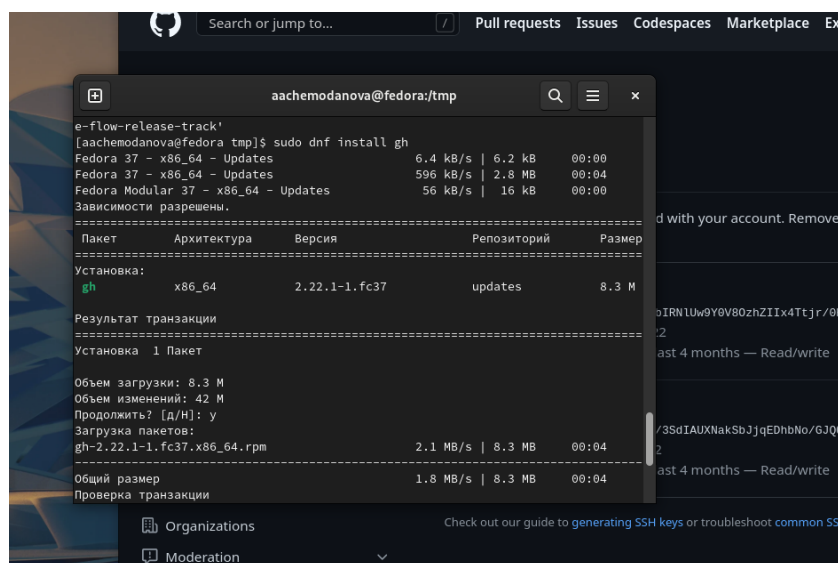


Рис. 4.3: Установка gh

Перейдем к первичной настройке параметров git. (рис. [4.4]).

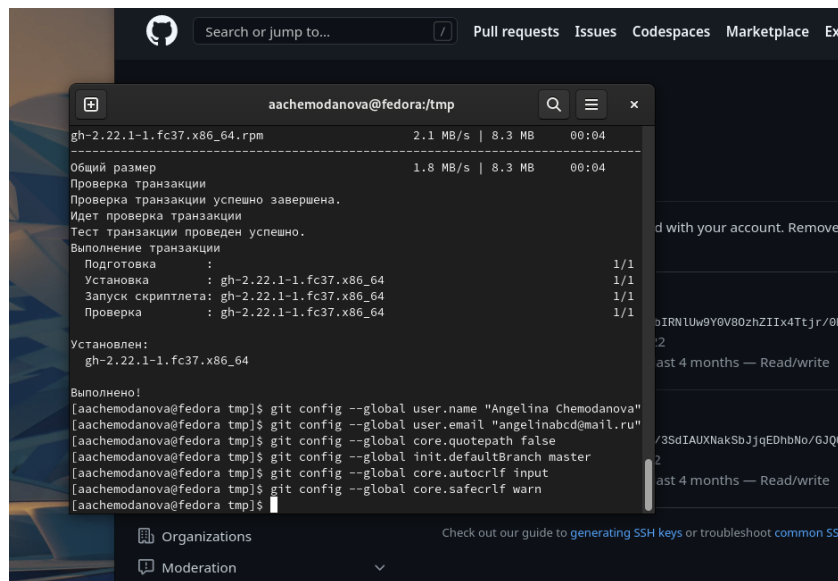


Рис. 4.4: Первичная настройка параметров git

Создаем ключ ssh по алгоритму rsa с ключем размером 4096 бит. (рис. [4.5]).

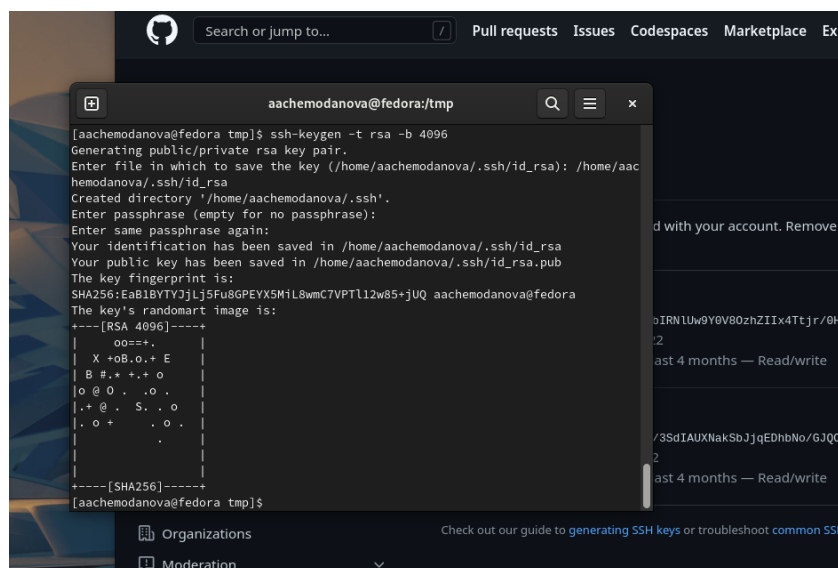
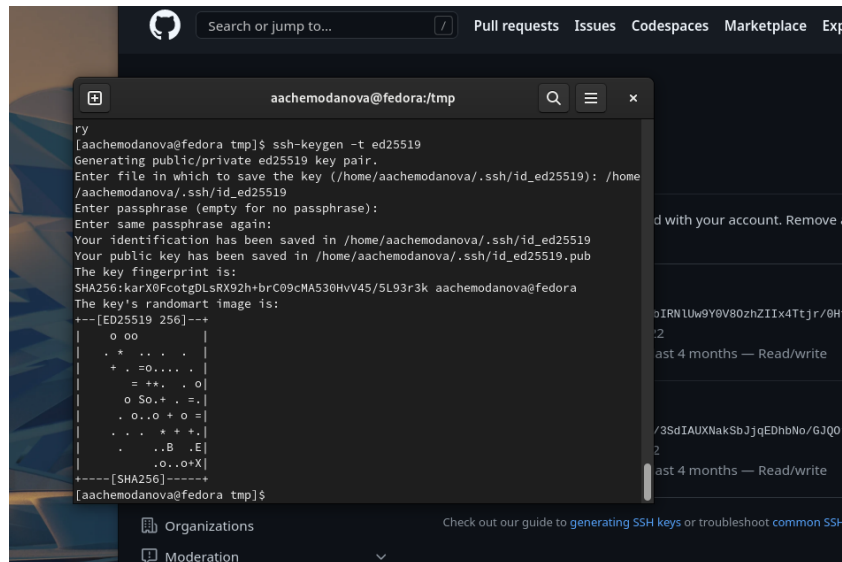


Рис. 4.5: Создание ключа ssh по алгоритму rsa с ключем размером 4096 бит

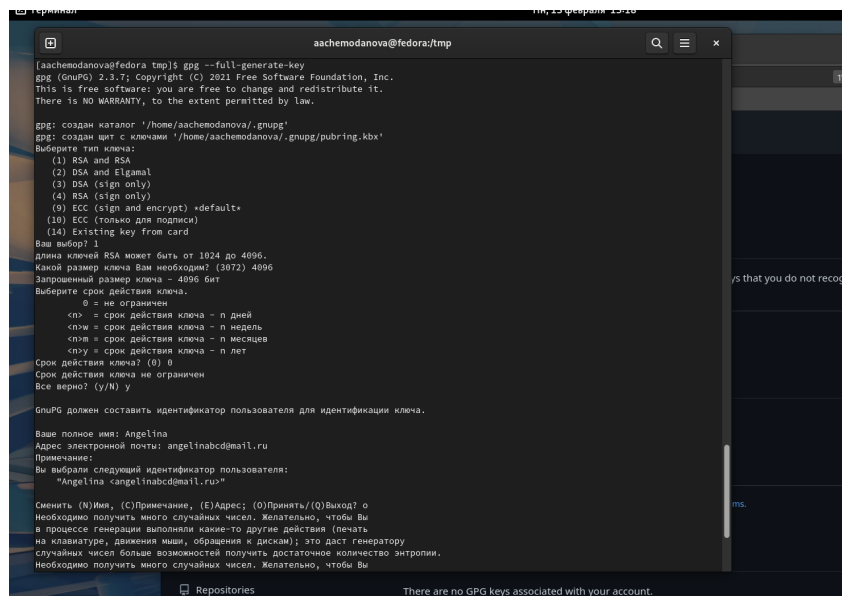
Создаем ключ ssh по алгоритму ed25519. (рис. [4.6]).



```
aachemodanova@fedora:tmp
ry
[aachemodanova@fedora tmp]$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/aachemodanova/.ssh/id_ed25519): /home
/aachemodanova/.ssh/id_ed25519
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/aachemodanova/.ssh/id_ed25519
Your public key has been saved in /home/aachemodanova/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:karX0FcotgDLsRX92h+brC09cMA530HvV45/5L93r3k aachemodanova@fedora
The key's randomart image is:
+--[ED25519 256]--+
  o oo
  . * . . .
  + . = 0 . . .
  = + * . . o
  o So . + =
  . o . o + o =
  . . . * + +
  . . .B .E
  . o . o + X
+-----[SHA256]-----
[aachemodanova@fedora tmp]$
```

Рис. 4.6: Создание ключа ssh по алгоритму ed25519

Создаем ключ gpg. Отвечаем на вопросы, которые заданы после введения команды. (рис. [4.7]).



```
aachemodanova@fedora:tmp
[aachemodanova@fedora tmp]$ gpg --full-generate-key
gpg (GnuPG) 2.3.7; 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 '/home/aachemodanova/.gnupg'
gpg: создан шифт с ключами '/home/aachemodanova/.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
<nw> = срок действия ключа - n неделя
<m> = срок действия ключа - n месяцев
<y> = срок действия ключа - n лет
Срок действия ключа? (0) 0
Срок действия ключа не ограничен
Все верно? (y/N) y

GnuPG должен составить идентификатор пользователя для идентификации ключа.
Ваше полное имя: Angelina
Адрес электронной почты: angelinabcd@mail.ru
Примечание:
Вы выбрали следующий идентификатор пользователя:
"Angelina <angelinabcd@mail.ru>"

Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход; 0
Необходимо получить много случайных чисел. Желательно, чтобы Вы
в процессе генерации выполняли какие-то другие действия (печатать
на клавиатуре, движения мыши, обращения к дискам); это даст генератору
случайных чисел больше возможностей получить достаточное количество энтропии.
Необходимо получить много случайных чисел. Желательно, чтобы Вы
```

Рис. 4.7: Создание ключа gpg

Выведем список ключей и добавим gpg-ключ на GitHub. (рис. [4.8]).

```

pub  rsa4096 2023-02-13 [SC]
     580CF76B32F06D59FA808E38B08977591B3CC46
uid      Angelina <angelinabcd@mail.ru>
sub  rsa4096 2023-02-13 [E]

[aachemodanova@fedora tmp]$ gpg --list-secret-keys --keyid-format LONG
bash: gpg: команда не найдена...
[aachemodanova@fedora tmp]$ gpg --list-secret-keys --keyid-format LONG
gpg: проверка таблицы доверия
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: глубина: 0 достоверных: 1 подписанных: 0 доверие: 0-, 0q, 0n, 0f, 1u
/home/aachemodanova/.gnupg/pubring.kbx
-----
sec  rsa4096/8B08977591B3CC46 2023-02-13 [SC]
     580CF76B32F06D59FA808E38B08977591B3CC46
uid      [ абсолотно ] Angelina <angelinabcd@mail.ru>
ssb  rsa4096/CC7EC82D462C3F8B 2023-02-13 [E]

[aachemodanova@fedora tmp]$

```

Рис. 4.8: Отпечаток приватного ключа

Добавляем ргр-ключ на гитхаб. (рис. [4.9]).

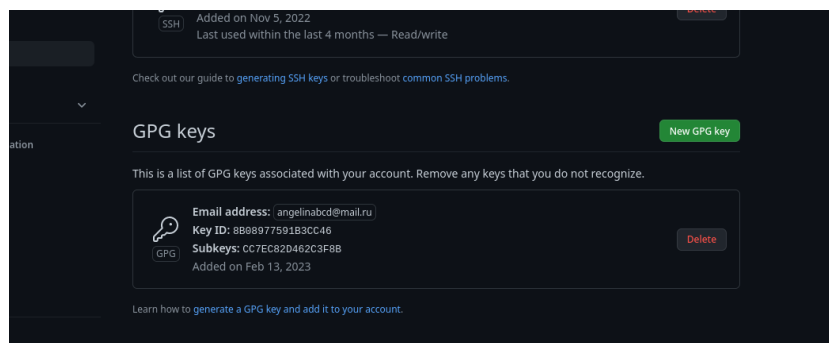


Рис. 4.9: Добавление ргр-ключ на гитхаб

Настраиваем автоматическую подпись коммитов git. (рис. [4.10]).

```

* Запрос данных...
* Проверка изменений...
* Установка пакетов...

[aachemodanova@fedora tmp]$ gpg --armor --export 8B08977591B3CC46 | xclip -sel clip
[aachemodanova@fedora tmp]$ git config --global user.signingkey 8B08977591B3CC46
git: «config» не является командой git. Смотрите «git --help».

Самые похожие команды:
config
[aachemodanova@fedora tmp]$ git config --global user.signingkey 8B08977591B3CC46
[aachemodanova@fedora tmp]$ git config --global commit.gpgsign true
[aachemodanova@fedora tmp]$ git config --global gpg.program $(which gpg2)
[aachemodanova@fedora tmp]$

```

Рис. 4.10: Настройка автоматических подписей коммитов git

Настройка gh и подтверждение авторизации. (рис. [4.11]).

Настройка каталога курса. (рис. [4.14]).

```
Определение изменений: 100% (40/40), готово.
Submodule path 'template/presentation': checked out 'b1be3800ee91f5809264cb755d316174540b753e'
Submodule path 'template/report': checked out '1d1b61dcac9c287a83917b2e3aef11a33b1e3b2'
[aachemodanova@fedora Операционные системы]$ cd ~/work/study/2022-2023/"Операционные системы"/os-intro
bash: cd: ~/work/study/2022-2023/Операционные системы/os-intro: Нет такого файла или каталога
[aachemodanova@fedora Операционные системы]$ cd ~/work/study/2022-2023/"Операционные системы"
[aachemodanova@fedora Операционные системы]$ cd ~/work/study/2022-2023/"Операционные системы"/os-intro
[aachemodanova@fedora os-intro]$ rm package.json
[aachemodanova@fedora os-intro]$ make COURSE=os-intro
[aachemodanova@fedora os-intro]$ git add .
[aachemodanova@fedora os-intro]$ git commit -am 'feat(main): make course structure'
(master 16fc273) feat(main): make course structure
360 files changed, 100326 insertions(+), 14 deletions(-)
create mode 100644 labs/README.md
create mode 100644 labs/README.ru.md
create mode 100644 labs/lab01/presentation/Makefile
create mode 100644 labs/lab01/presentation/image/kulyabov.jpg
create mode 100644 labs/lab01/presentation/presentation.md
create mode 100644 labs/lab01/report/Makefile
create mode 100644 labs/lab01/report/bib/cite.bib
create mode 100644 labs/lab01/report/image/placeimg_800_600_tech.jpg
create mode 100644 labs/lab01/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 100755 labs/lab01/report/pandoc/filters/pandoc_eqnos.py
create mode 100755 labs/lab01/report/pandoc/filters/pandoc_fignos.py
README.git-flow.md Initial commit 1 hour ago Publish
```

Рис. 4.14: Настройка каталога курса

(рис. [4.15]).

```
create mode 100755 project-personal/stage6/report/pandoc/filters/pandoc_eqnos.py
create mode 100755 project-personal/stage6/report/pandoc/filters/pandoc_fignos.py
create mode 100755 project-personal/stage6/report/pandoc/filters/pandoc_secnos.py
create mode 100755 project-personal/stage6/report/pandoc/filters/pandoc_tablenos.py
create mode 100644 project-personal/stage6/report/pandoc/filters/pandocxnos/_init_.py
create mode 100644 project-personal/stage6/report/pandoc/filters/pandocxnos/core.py
create mode 100644 project-personal/stage6/report/pandoc/filters/pandocxnos/main.py
create mode 100644 project-personal/stage6/report/pandoc/filters/pandocxnos/pandocattributes.py
create mode 100644 project-personal/stage6/report/report.md
[aachemodanova@fedora os-intro]$ git push
Перечисление объектов: 38, готово.
Подсчет объектов: 100% (38/38), готово.
При сжатии изменений используется до 2 потоков
Сжатие объектов: 100% (38/38), готово.
Запись объектов: 100% (37/37), 343.00 Киб | 2.43 Миб/с, готово.
Всего 37 (изменений 4), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To github.com:aachemodanova/study_2022-2023_os-intro.git
c55e57d..16fc273 master -> master
[aachemodanova@fedora os-intro]$
README.git-flow.md Initial commit 1 hour ago
```

Рис. 4.15: Настройка каталога курса

5 Выводы

Мы изучили идеологию и применение средств контроля версий, а также освоили умения по работе с git.

6 Контрольные вопросы

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

Система контроля версий (VCS) - программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое. Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов.

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

Хранилище – репозиторий - место хранения всех версий и служебной информации. Commit - это команда для записи индексированных изменений в репозиторий. История – место, где сохраняются все коммиты, по которым можно посмотреть данные о коммитах. Рабочая копия – текущее состояние файлов проекта, основанное на версии, загруженной из хранилища.

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

Централизованные системы – это системы, в которых одно основное хранилище всего проекта, и каждый пользователь копирует необходимые ему файлы, изменяет и вставляет обратно. Пример – Subversion. Децентрализованные системы – система, в которой каждый пользователь имеет свой вариант репозитория и есть возможность добавлять и забирать изменения из репозитория. Пример – Git.

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

В рабочей копии, которую исправляет человек, появляются правки, которые отправляются в хранилище на каждом из этапов. То есть в правки в рабочей копии появляются, только если человек делает их (отправляет их на сервер) и никак по-другому.

5. Опишите порядок работы с общим хранилищем VCS?

Если хранилище общее, то в рабочую копию каждого, кто работает над проектом, приходят изменения, отправленные на сервер одним из команды. Рабочая правка каждого может изменяться вне зависимости от того, делает ли конкретный человек правки или нет.

6. Каковы основные задачи, решаемые инструментальным средством git?

У Git две основных задачи: первая — хранить информацию обо всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом.

7. Назовите и дайте краткую характеристику командам 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 изменения: `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` –
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'` – 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 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. Приведите примеры использования при работе с локальным и удалённым репозиториями.

Работа с удалённым репозиторием: `git remote` – просмотр списка настроенных удалённых репозиториев.

Работа с локальным репозиторием: `git status` – выводит информацию обо всех изменениях, внесенных в дерево директорий проекта по сравнению с последним коммитом рабочей ветки

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

Ветка (англ. branch) — это последовательность коммитов, в которой ведётся параллельная разработка какого-либо функционала. Ветки нужны, чтобы несколько программистов могли вести работу над одним и тем же проектом или даже

файлом одновременно, при этом не мешая друг другу. Кроме того, ветки используются для тестирования экспериментальных функций: чтобы не повредить основному проекту, создается новая ветка специально для экспериментов.

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

Игнорируемые файлы — это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты. В Git нет специальной команды для указания игнорируемых файлов: вместо этого необходимо вручную отредактировать файл. Временно игнорировать изменения в файле можно командой `git update-index--assumeunchanged`