Лабораторная 2

Операционные системы

Прокопьева Марина Евгеньевна НБИбд-02-23

Содержание

# 1 Цель работы

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

# 2 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.

# 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 просмотреть 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

**Работа с локальным репозиторием**

Создадим локальный репозиторий.  
  
Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория:  
  
git config --global user.name "Имя Фамилия"  
git config --global user.email "work@mail"  
  
Настроим utf-8 в выводе сообщений git:  
  
git config --global quotepath false  
  
Для инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial, необходимо ввести в командной строке:  
  
cd   
mkdir tutorial  
cd tutorial  
git init  
  
После это в каталоге tutorial появится каталог .git, в котором будет храниться история изменений.  
  
Создадим тестовый текстовый файл hello.txt и добавим его в локальный репозиторий:  
  
echo 'hello world' > hello.txt  
git add hello.txt  
git commit -am 'Новый файл'  
  
Воспользуемся командой status для просмотра изменений в рабочем каталоге, сделанных с момента последней ревизии:  
  
git status  
  
Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл .gitignore с помощью сервисов. Для этого сначала нужно получить список имеющихся шаблонов:  
  
curl -L -s https://www.gitignore.io/api/list  
  
Затем скачать шаблон, например, для C и C++  
  
curl -L -s https://www.gitignore.io/api/c >> .gitignore  
curl -L -s https://www.gitignore.io/api/c++ >> .gitignore

**Работа с сервером репозиториев**

Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый):  
  
ssh-keygen -C "Имя Фамилия <work@mail>"  
  
Ключи сохраняться в каталоге ~/.ssh/.  
Существует несколько доступных серверов репозиториев с возможностью бесплатного размещения данных. Например, https://github.com/.  
Для работы с ним необходимо сначала завести на сайте https://github.com/ учётную запись. Затем необходимо загрузить сгенерённый нами ранее открытый ключ.  
Для этого зайти на сайт https://github.com/ под своей учётной записью и перейти в меню GitHub setting.  
  
После этого выбрать в боковом меню GitHub setting>SSH-ключи и нажать кнопку Добавить ключ. Скопировав из локальной консоли ключ в буфер обмена:  
  
cat ~/.ssh/id\_rsa.pub | xclip -sel clip  
  
Вставляем ключ в появившееся на сайте поле.  
  
После этого можно создать на сайте репозиторий, выбрав в меню , дать ему название и сделать общедоступным (публичным).  
  
Для загрузки репозитория из локального каталога на сервер выполняем следующие команды:  
  
git remote add origin   
 ssh://git@github.com/<username>/<reponame>.git  
git push -u origin master  
  
Далее на локальном компьютере можно выполнять стандартные процедуры для работы с git при наличии центрального репозитория.

**Базовая настройка git**

**Первичная настройка параметров git**

Зададим имя и email владельца репозитория:  
  
git config --global user.name "Name Surname"  
git config --global user.email "work@mail"  
  
Настроим utf-8 в выводе сообщений git:  
  
git config --global core.quotepath false  
  
Настройте верификацию и подписание коммитов git.  
  
Зададим имя начальной ветки (будем называть её master):  
  
git config --global init.defaultBranch master

**Учёт переносов строк**

В разных операционных системах приняты разные символы для перевода строк:  
 Windows: \r\n (CR и LF);  
 Unix: \n (LF);  
 Mac: \r (CR).  
  
Посмотреть значения переносов строк в репозитории можно командой:  
  
git ls-files --eol  
  
Параметр autocrlf  
 Настройка core.autocrlf предназначена для того, чтобы в главном репозитории все переводы строк текстовых файлах были одинаковы.  
 Настройка core.autocrlf с параметрами true и input делает все переводы строк текстовых файлов в главном репозитории одинаковыми.  
 core.autocrlf true: конвертация CRLF->LF при коммите и обратно LF->CRLF при выгрузке кода из репозитория на файловую систему (обычно используется в Windows).  
 core.autocrlf input: конвертация CRLF->LF только при коммитах (используются в MacOS/Linux).  
  
 Варианты конвертации  
   
   
 \*\*Создание ключа ssh\*\*  
 \*\*Общая информация\*\*  
   
  
Алгоритмы шифрования ssh  
  
 Аутентификация  
  
 В SSH поддерживается четыре алгоритма аутентификации по открытым ключам:  
 DSA:  
 размер ключей DSA не может превышать 1024, его следует отключить;  
 RSA:  
 следует создавать ключ большого размера: 4096 бит;  
 ECDSA:  
 ECDSA завязан на технологиях NIST, его следует отключить;  
 Ed25519:  
 используется пока не везде.  
  
 Симметричные шифры  
 Из 15 поддерживаемых в SSH алгоритмов симметричного шифрования, безопасными можно считать:  
 chacha20-poly1305;  
 aes\*-ctr;  
 aes\*-gcm.  
 Шифры 3des-cbc и arcfour потенциально уязвимы в силу использования DES и RC4.  
 Шифр cast128-cbc применяет слишком короткий размер блока (64 бит).  
  
 Обмен ключами  
 Применяемые в SSH методы обмена ключей DH (Diffie-Hellman) и ECDH (Elliptic Curve Diffie-Hellman) можно считать безопасными.  
 Из 8 поддерживаемых в SSH протоколов обмена ключами вызывают подозрения три, основанные на рекомендациях NIST:  
 ecdh-sha2-nistp256;  
 ecdh-sha2-nistp384;  
 ecdh-sha2-nistp521.  
 Не стоит использовать протоколы, основанные на SHA1.  
  
Файлы ssh-ключей  
 По умолчанию пользовательские ssh-ключи сохраняются в каталоге ~/.ssh в домашнем каталоге пользователя.  
 Убедитесь, что у вас ещё нет ключа.

Файлы закрытых ключей имеют названия типа id\_ (например, id\_dsa, id\_rsa).

По умолчанию закрытые ключи имеют имена:

id\_dsa id\_ecdsa id\_ed25519 id\_rsa

Открытые ключи имеют дополнительные расширения .pub.

По умолчанию публичные ключи имеют имена:

id\_dsa.pub id\_ecdsa.pub id\_ed25519.pub id\_rsa.pub

При создании ключа команда попросит ввести любую ключевую фразу для более надёжной защиты вашего пароля. Можно пропустить этот этап, нажав Enter.

Сменить пароль на ключ можно с помощью команды:

ssh-keygen -p

**Создание ключа ssh**

Ключ ssh создаётся командой:  
  
ssh-keygen -t <алгоритм>  
  
Создайте ключи:

по алгоритму rsa с ключём размером 4096 бит:

ssh-keygen -t rsa -b 4096

по алгоритму ed25519:

ssh-keygen -t ed25519

При создании ключа команда попросит ввести любую ключевую фразу для более надёжной защиты вашего пароля. Можно пропустить этот этап, нажав Enter.  
  
Сменить пароль на ключ можно с помощью команды:  
  
ssh-keygen -p

**Добавление SSH-ключа в учётную запись GitHub**

Скопируйте созданный SSH-ключ в буфер обмена командой:  
  
xclip -i < ~/.ssh/id\_ed25519.pub  
  
Откройте настройки своего аккаунта на GitHub и перейдем в раздел SSH and GPC keys.  
Нажмите кнопку ew SSH key.  
Добавьте в поле Title название этого ключа, например, ed25519@hostname.  
Вставьте из буфера обмена в поле Key ключ.  
Нажмите кнопку Add SSH key.

**Верификация коммитов с помощью PGP**

**Общая информация**

Коммиты имеют следующие свойства:

author (автор) — контрибьютор, выполнивший работу (указывается для справки); committer (коммитер) — пользователь, который закоммитил изменения. Эти свойства можно переопределить при совершении коммита. Авторство коммита можно подделать. В git есть функция подписи коммитов. Для подписывания коммитов используется технология PGP (см. Работа с PGP). Подпись коммита позволяет удостовериться в том, кто является коммитером. Авторство не проверяется.

**Создание ключа**

Генерируем ключ  
  
gpg --full-generate-key  
  
Из предложенных опций выбираем:

тип RSA and RSA; размер 4096; выберите срок действия; значение по умолчанию — 0 (срок действия не истекает никогда). GPG запросит личную информацию, которая сохранится в ключе: Имя (не менее 5 символов). Адрес электронной почты. При вводе email убедитесь, что он соответствует адресу, используемому на GitHub. Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить это поле пустым.

**Экспорт ключа**

Выводим список ключей и копируем отпечаток приватного ключа:  
  
gpg --list-secret-keys --keyid-format LONG  
  
Отпечаток ключа — это последовательность байтов, используемая для идентификации более длинного, по сравнению с самим отпечатком ключа.  
  
Формат строки:  
  
sec Алгоритм/Отпечаток\_ключа Дата\_создания [Флаги] [Годен\_до]

ID\_ключа

Экспортируем ключ в формате ASCII по его отпечатку:  
  
gpg --armor --export <PGP Fingerprint>

**Добавление PGP ключа в GitHub**

Копируем ключ и добавляем его в настройках профиля на GitHub (или GitLab).  
  
Cкопируйте ваш сгенерированный PGP ключ в буфер обмена:  
  
gpg --armor --export <PGP Fingerprint> | xclip -sel clip  
  
Перейдите в настройки GitHub (https://github.com/settings/keys), нажмите на кнопку New GPG key и вставьте полученный ключ в поле ввода.

**Подписывание коммитов git**

Подпись коммитов при работе через терминал:  
  
git commit -a -S -m 'your commit message'  
  
Флаг -S означает создание подписанного коммита. При этом может потребоваться ввод кодовой фразы, заданной при генерации GPG-ключа.

**Настройка автоматических подписей коммитов git**

Используя введёный email, укажите Git применять его при подписи коммитов:  
  
git config --global user.signingkey <PGP Fingerprint>  
git config --global commit.gpgsign true  
git config --global gpg.program $(which gpg2)

**Проверка коммитов в Git** **Режим бдительности (vigilant mode)**

На GitHub есть настройка vigilant mode.  
Все неподписанные коммиты будут явно помечены как Unverified.  
Включается это в настройках в разделе SSH and GPG keys. Установите метку на Flag unsigned commits as unverified.

# 4 Выполнение лабораторной работы

*Зададим имя и email владельца репозитория:*

настройка

Рис. 1: настройка

*Зададим имя начальной ветки (будем называть её master), Параметр autocrlf, Параметр safecrlf:*

настройка

Рис. 2: настройка

*Создаю ключи ssh*

по алгоритму rsa с ключём размером 4096 бит:

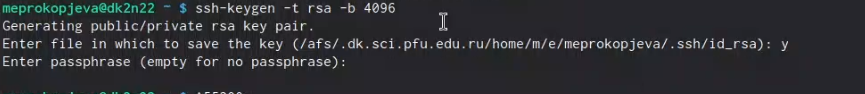


Рис. 3: ключ ssh

по алгоритму ed25519:

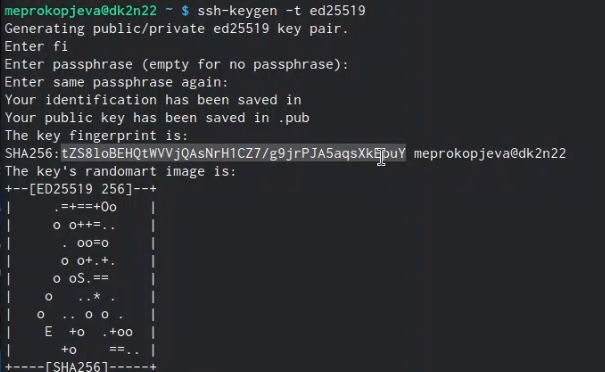


Рис. 4: ключ ssh

*Создаю ключи pgp*

Генерируем ключ

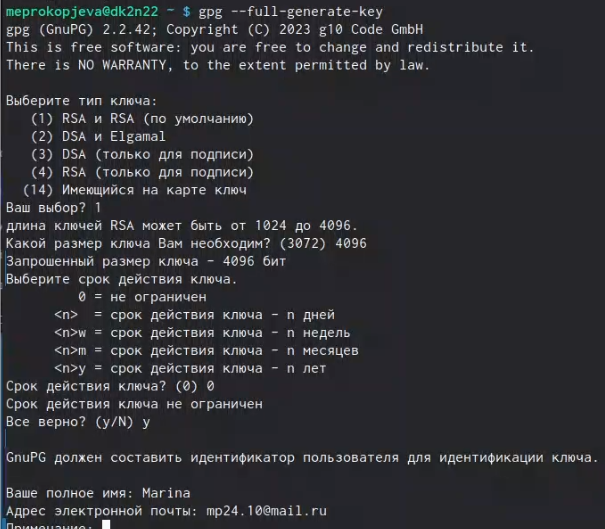


Рис. 5: ключ pgp

Из предложенных опций выбираем:

тип RSA and RSA;  
размер 4096;  
выберала срок действия; значение по умолчанию — 0 (срок действия не истекает никогда).

GPG запросит личную информацию, которая сохранится в ключе:

Имя (не менее 5 символов).  
Адрес электронной почты.  
При вводе email убедилась, что он соответствует адресу, используемому на GitHub.  
Комментарий.

*Добавление PGP ключа в GitHub*

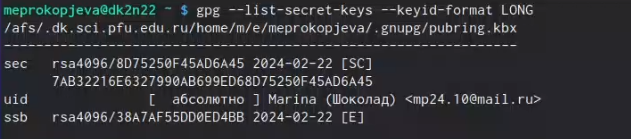


Рис. 6: ключ pgp

Cкопируйте ваш сгенерированный PGP ключ в буфер обмена:

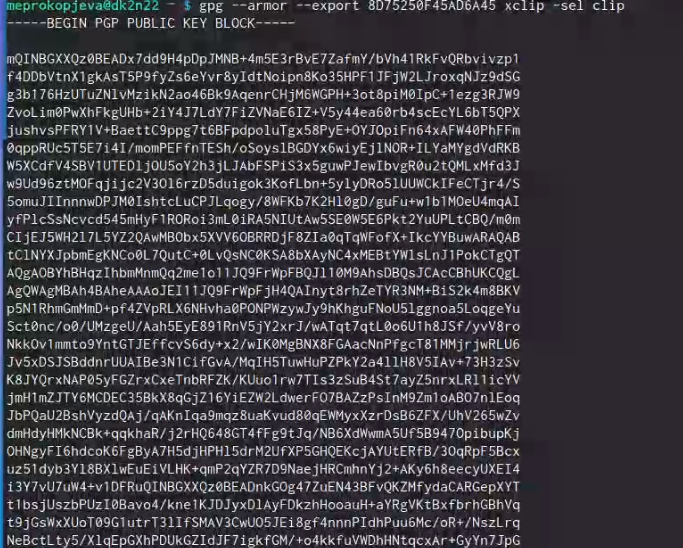


Рис. 7: ключ pgp

*Настройка автоматических подписей коммитов git*

настройка

Рис. 8: настройка

*Настройка gh*

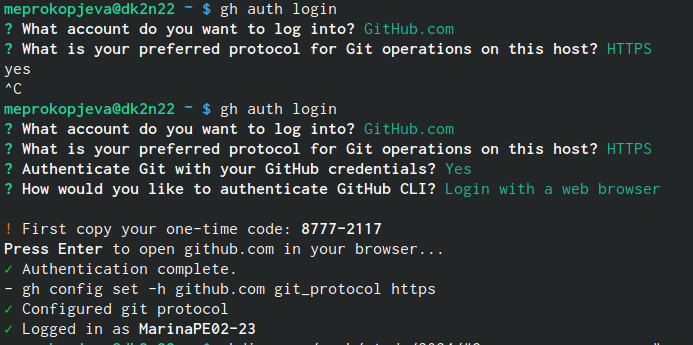


Рис. 9: настройка

*Сознание репозитория курса на основе шаблона*

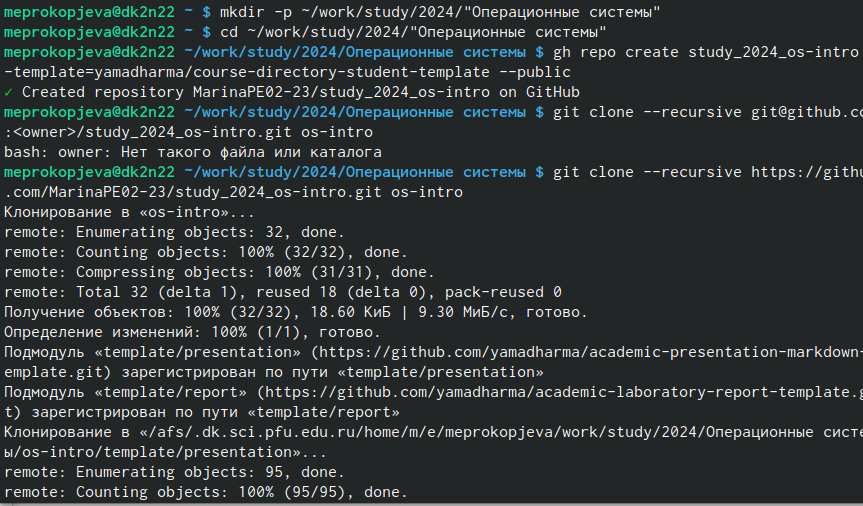


Рис. 10: создание

*Настройка каталога курса*

Перешла в каталог курса:

настройка

Рис. 11: настройка

Удалить лишние файлы:

настройка

Рис. 12: настройка

Создала необходимые каталоги:

настройка

Рис. 13: настройка

настройка

Рис. 14: настройка

Отправила файлы на сервер

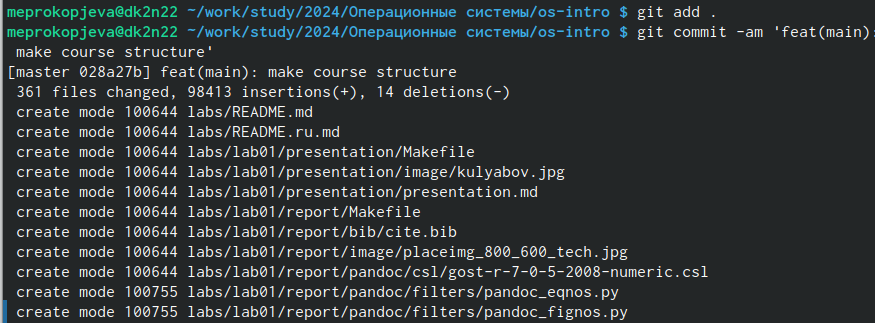


Рис. 15: настройка

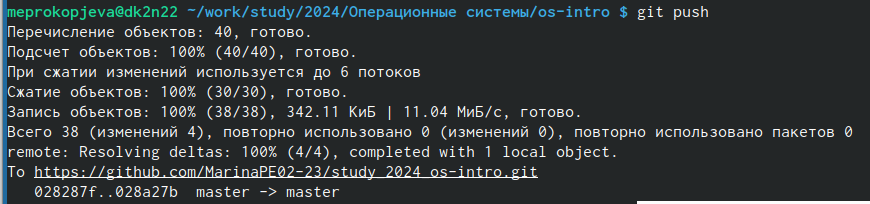


Рис. 16: настройка

# 5 Выводы

Изучили идеологию и применение средств контрося вериский и освоили умения по работе с git.

# 6 Контрольные вопросы

1. Системы контроля версий (VCS) - программное обеспечение для облегчения работы с изменяющейся информацией. Они позволяют хранить несколько версий изменяющейся информации, одного и того же документа, может предоставить доступ к более ранним версиям документа. Используется для работы нескольких человек над проектом, позволяет посмотреть, кто и когда внес какое-либо изменение и т. д. VCS ррименяю1тся для: Хранения понлой истории изменений, сохранения причин всех изменений, поиска причин изменений и совершивших изменение, совместной работы над проектами.
2. Хранилище – репозиторий, хранилище версий, в нем хранятся все документы, включая историю их изменения и прочей служебной информацией. commit – отслеживание изменений, сохраняет разницу в изменениях. История – хранит все изменения в проекте и позволяет при необходимости вернуться/обратиться к нужным данным. Рабочая копия – копия проекта, основанная на версии из хранилища, чаще всего последней версии.
3. Централизованные VCS (например: CVS, TFS, AccuRev) – одно основное хранилище всего проекта. Каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет, затем добавляет изменения обратно в хранилище. Децентрализованные VCS (например: Git, Bazaar) – у каждого пользователя свой вариант репозитория (возможно несколько вариантов), есть возможность добавлять и забирать изменения из любого репозитория. В отличие от классических, в распределенных (децентралиованных) системах контроля версий центральный репозиторий не является обязательным.
4. Сначала создается и подключается удаленный репозиторий, затем по мере изменения проекта эти изменения отправляются на сервер.
5. Участник проекта перед началом работы получает нужную ему версию проекта в хранилище, с помощью определенных команд, после внесения изменений пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются. К ним можно вернуться в любой момент.
6. Хранение информации о всех изменениях в вашем коде, обеспечение удобства командной работы над кодом.
7. Создание основного дерева репозитория: git init Получение обновлений (изменений) текущего дерева из центрального репозитория: git pull Отправка всех произведённых изменений локального дерева в центральный репозиторий: git push Просмотр списка изменённых файлов в текущей директории: git status Просмотр текущих изменений: git diff Сохранение текущих изменений: добавить все изменённые и/или созданные файлы и/или каталоги: git add . добавить конкретные изменённые и/или созданные файлы и/или каталоги: git add имена\_файлов удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm имена\_файлов Сохранение добавленных изменений: сохранить все добавленные изменения и все изменённые файлы: git commit -am ‘Описание коммита’ сохранить добавленные изменения с внесением комментария через встроенный редактор: git commit создание новой ветки, базирующейся на текущей: git checkout -b имя\_ветки переключение на некоторую ветку: git checkout имя\_ветки (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) отправка изменений конкретной ветки в центральный репозиторий: git push origin имя\_ветки слияние ветки с текущим деревом: git merge –no-ff имя\_ветки Удаление ветки: удаление локальной уже слитой с основным деревом ветки: git branch -d имя\_ветки принудительное удаление локальной ветки: git branch -D имя\_ветки удаление ветки с центрального репозитория: git push origin :имя\_ветки
8. git push -all отправляем из локального репозитория все сохраненные изменения в центральный репозиторий, предварительно создав локальный репозиторий и сделав предварительную конфигурацию.
9. Ветвление - один из параллельных участков в одном хранилище, исходящих из одной версии, обычно есть главная ветка. Между ветками, т. е. их концами возможно их слияние. Используются для разработки новых функций.
10. Во время работы над проектом могут создаваться файлы, которые не следуют добавлять в репозиторий. Например, временные файлы. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл .gitignore с помощью сервисов.