Лабораторная работа 2

Первоначальная настройка git

Руденко Михаил Андреевич

Содержание

Список иллюстраций

Список таблиц

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

3.1 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.

3.2 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

3.3 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

3.4 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

4 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//.git git push -u origin master

Далее на локальном компьютере можно выполнять стандартные процедуры для работы с git при наличии центрального репозитория.

4.1 Базовая настройка git

4.2 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 Таблица 1.: Варианты конвертации для разных значений параметра core.autocrlf core.autocrlf false input true git commit LF -> LF LF -> LF LF -> CRLF CR -> CR CR -> CR CR -> CR CRLF -> CRLF CRLF -> LF CRLF -> CRLF git checkout LF -> LF LF -> LF LF -> CRLF CR -> CR CR -> CR CR -> CR CRLF -> CRLF

CRLF -> CRLF CRLF -> CRLF

Установка параметра:

Для Windows

git config –global core.autocrlf true

Для Linux

git config –global core.autocrlf input

Параметр safecrlf Настройка core.safecrlf предназначена для проверки, является ли окончаний строк обратимым для текущей настройки core.autocrlf. core.safecrlf true: запрещается необратимое преобразование lf<->crlf. Полезно, когда существуют бинарные файлы, похожие на текстовые файлы. core.safecrlf warn: печать предупреждения, но коммиты с необратимым переходом принимаются.

Установка параметра:

git config –global core.safecrlf warn

4.3 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”

Описываются проведённые действия, в качестве иллюстрации даётся ссылка на иллюстрацию (рис. 5.1).

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.

4.4 Верификация коммитов с помощью PGP

Как настроить PGP-подпись коммитов с помощью gpg.

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

Добавление PGP ключа в GitHub

Копируем ключ и добавляем его в настройках профиля на GitHub (или GitLab).

Скопируйте ваш сгенерированный PGP ключ в буфер обмена:

gpg –armor –export | 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.

4.5 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.

4.6 Настройка автоматических подписей коммитов git

Используя введёный email, укажите Git применять его при подписи коммитов:

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

Проверка коммитов в Git

GitHub и GitLab будут показывать значок Verified рядом с вашими новыми коммитами.

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.

5 Выполнение лабораторной работы

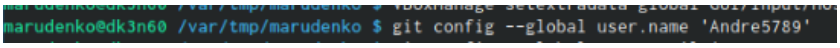


Рис. 5.1: Зададим имя

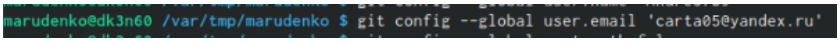


Рис. 5.2: Зададим email владельца репозитория:

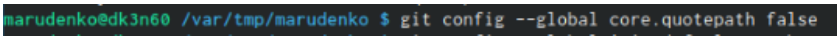


Рис. 5.3: Настроим utf-8 в выводе сообщений git:

Рис. 5.4: Зададим имя начальной ветки (будем называть её master):

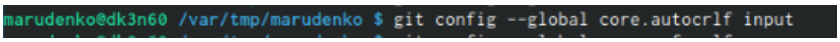


Рис. 5.5: Настройка core.autocrlf предназначена для того, чтобы в главном репозитории все переводы строк текстовых файлах были одинаковы.Настройка core.autocrlf с параметрами true и input делает все переводы строк текстовых файлов в главном репозитории одинаковыми.core.autocrlf true: конвертация CRLF->LF 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

```
marudenko@dk3n60 /var/tmp/marudenko $ git config --global core.safecrlf warn
```

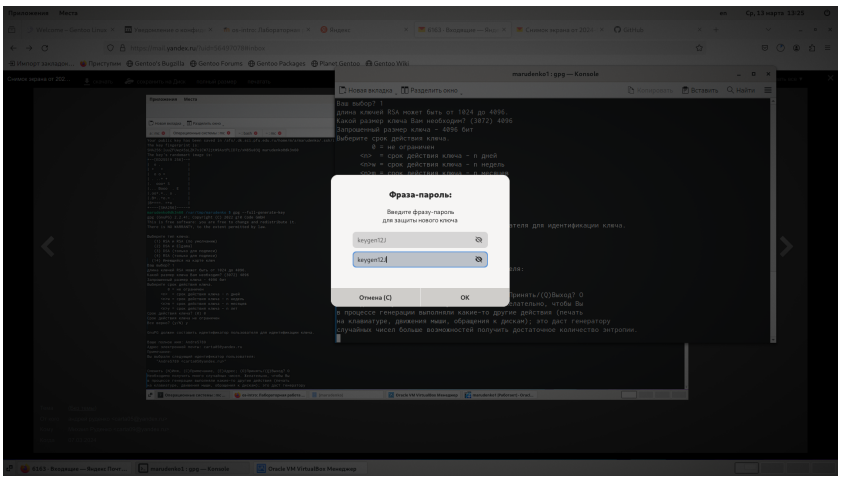
Рис. 5.6: Настройка core.safecrlf предназначена для проверки, является ли окончаний строк обратимым для текущей настройки core.autocrlf.core.safecrlf true: запрещается необратимое преобразование lf<->crlf. Полезно, когда существуют бинарные файлы, похожие на текстовые файлы.core.safecrlf warn: печать предупреждения, но коммиты с необратимым переходом принимаются. Установка параметра:git config –global core.safecrlf warn

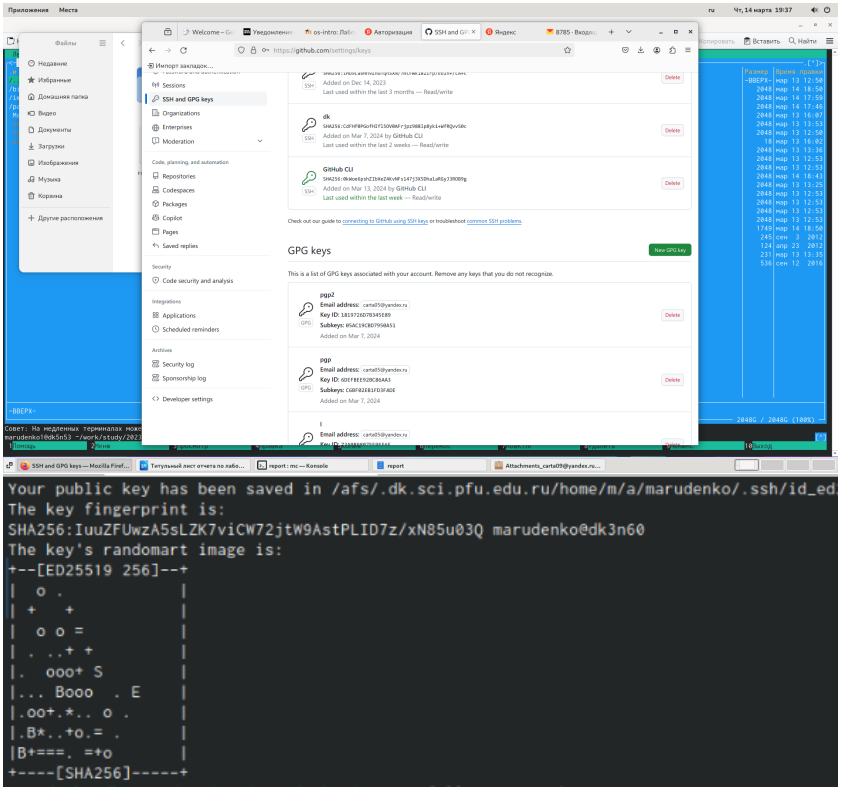
```
marudenko@dk3n60 /var/tmp/marudenko $ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/m/a/marudenko/.ssh/id_rsa):
/afs/.dk.sci.pfu.edu.ru/home/m/a/marudenko/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /afs/.dk.sci.pfu.edu.ru/home/m/a/marudenko/.ssh/id_rsa
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/m/a/marudenko/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:CdFHf0PGofHIfl5OV0AFrjpz988lp8yk1+WfRQvvS0c marudenko@dk3n60
The key's randomart image is:
+---[RSA 4096]----+
|       .. .. .oB+.|
|        .. .o O.. |
|       .  .  = = .|
|        . . . o ..|
|         S   o.. E|
|            . oo*+|
|           + . =*B|
|            + ***B|
|             ..++O|
+----[SHA256]-----+
```

Рис. 5.7: Создайте ключи:по алгоритму rsa с ключём размером 4096 бит:

```
marudenko@dk3n60 /var/tmp/marudenko $ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/m/a/marudenko/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /afs/.dk.sci.pfu.edu.ru/home/m/a/marudenko/.ssh/id_ed25519
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/m/a/marudenko/.ssh/id_ed25519.pub
```

Рис. 5.8: по алгоритму ed25519:ssh-keygen -t ed25519




GPG keys
New GPG key
SSH and GPG keys
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/m/a/marudenko/.ssh/id_ed
The key fingerprint is:
SHA256:IuuZFUwzA5sLZK7viCW72jtW9AstPLID7z/xN85u03Q marudenko@dk3n60
The key's randomart image is:
+--[ED25519 256]--+
| o . |
| +   + |
| o o = |
| . ..+ + |
|. ooo+ S |
|... Booo . E |
|.oo+.*.. o . |
|.B*..+o.= . |
|B+===. =+o |
+----[SHA256]-----+

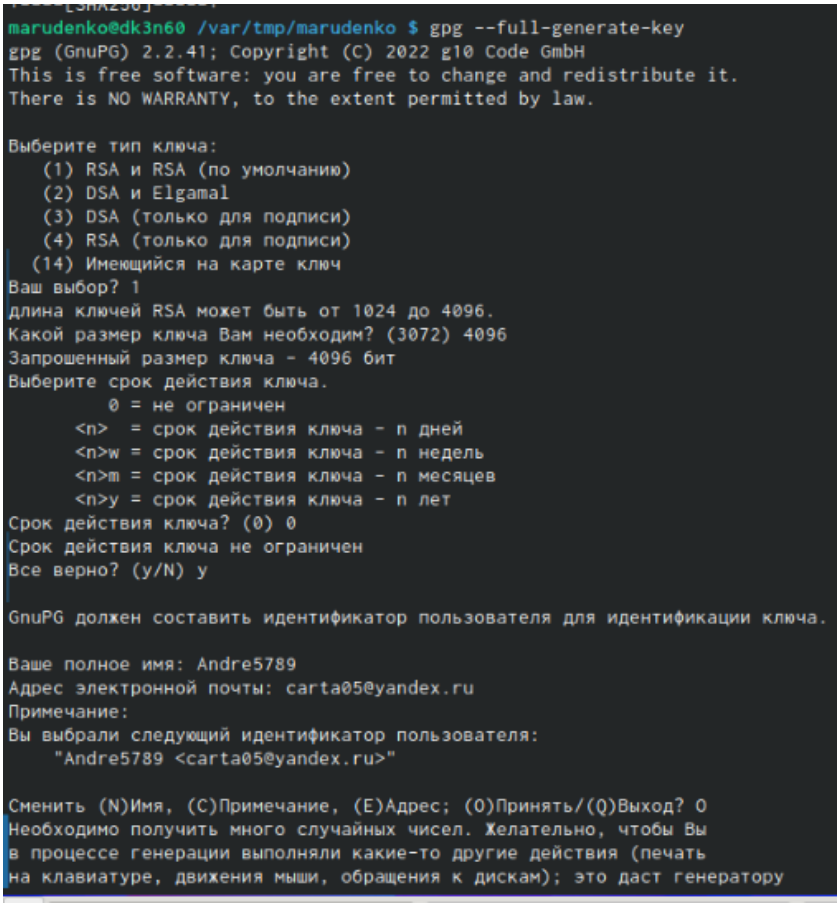
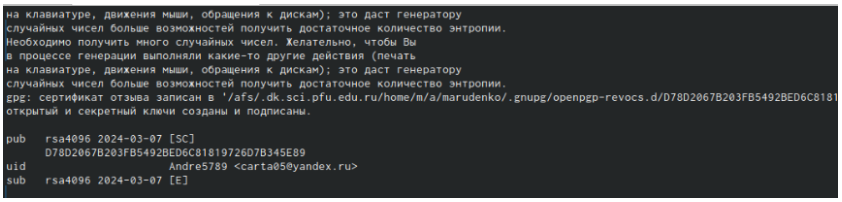


Рис. 5.9: Генерируем ключ gpg –full-generate-key



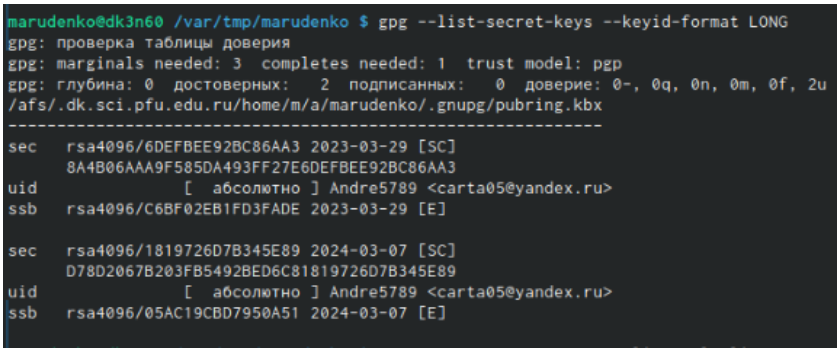


Рис. 5.10: Выводим список ключей и копируем отпечаток приватного ключа:

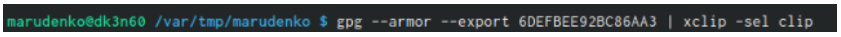


Рис. 5.11: Копируем ключ и добавляем его в настройках профиля на GitHub (или GitLab).Скопируйте ваш сгенерированный PGP ключ в буфер обмена:

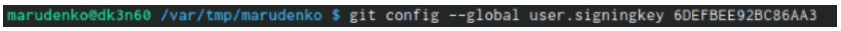
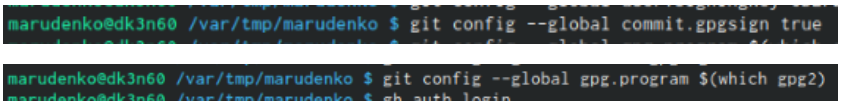


Рис. 5.12: Используя введёный email, укажите Git применять его при подписи коммитов:



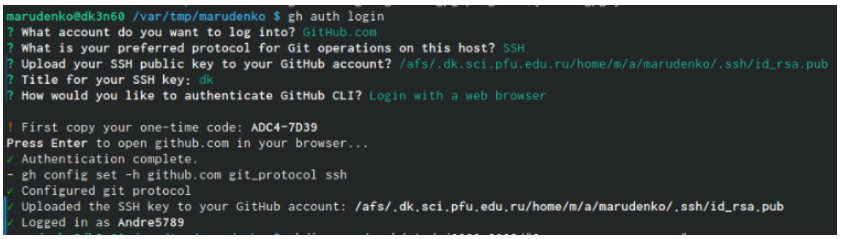


Рис. 5.13: Для начала необходимо авторизоваться

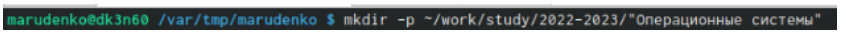
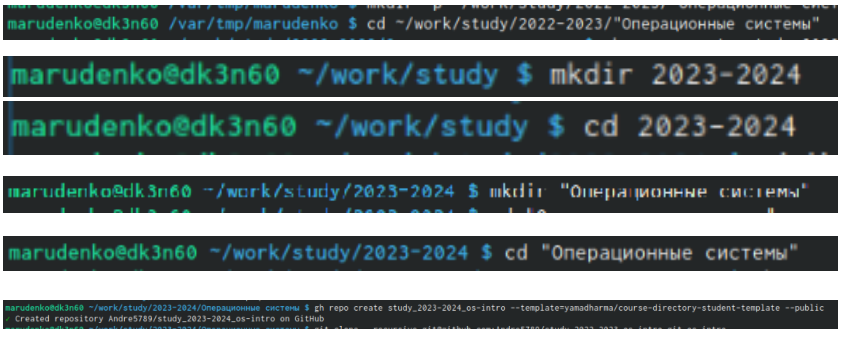


Рис. 5.14: Необходимо создать шаблон рабочего пространства (см. Рабочее пространство для лабораторной работы).Например, для 2022–2023 учебного года и предмета «Операционные системы» (код предмета os-intro) создание репозитория примет следующий вид:



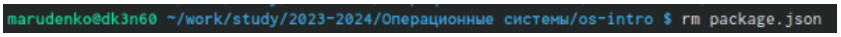


Рис. 5.15: удалим файл

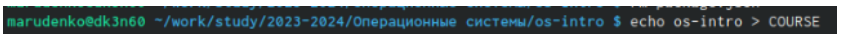


Рис. 5.16: Создайте необходимые каталоги:

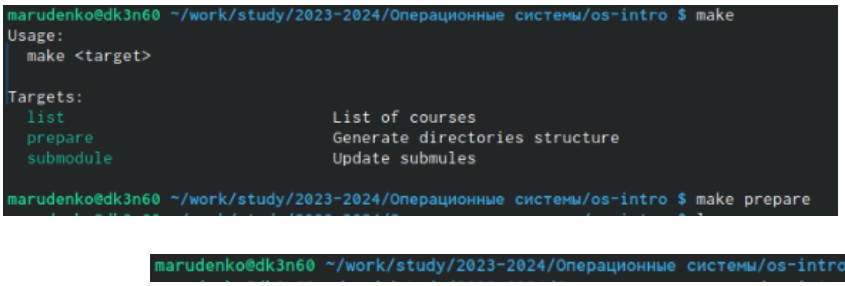
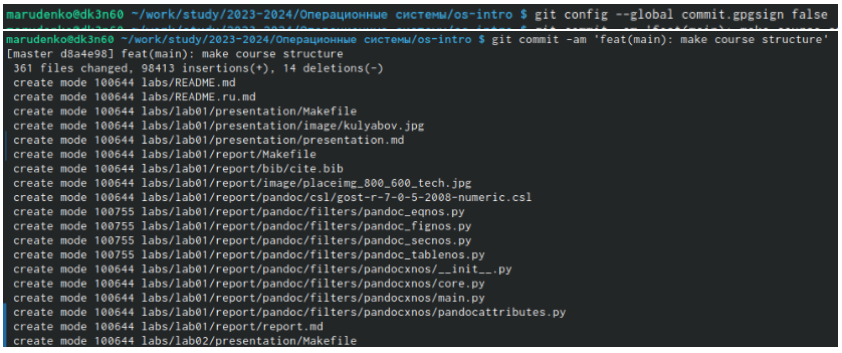


Рис. 5.17: Отправьте файлы на сервер:



```
 create mode 100755 project-personal/stage4/report/pandoc/filters/pandoc_eqnos.py
 create mode 100755 project-personal/stage4/report/pandoc/filters/pandoc_fignos.py
 create mode 100755 project-personal/stage4/report/pandoc/filters/pandoc_secnos.py
 create mode 100755 project-personal/stage4/report/pandoc/filters/pandoc_tablenos.py
 create mode 100644 project-personal/stage4/report/pandoc/filters/pandocxnos/__init__.py
 create mode 100644 project-personal/stage4/report/pandoc/filters/pandocxnos/core.py
 create mode 100644 project-personal/stage4/report/pandoc/filters/pandocxnos/main.py
 create mode 100644 project-personal/stage4/report/pandoc/filters/pandocxnos/pandocattributes.py
 create mode 100644 project-personal/stage4/report/report.md
 create mode 100644 project-personal/stage5/presentation/Makefile
 create mode 100644 project-personal/stage5/presentation/image/kulyabov.jpg
 create mode 100644 project-personal/stage5/presentation/presentation.md
 create mode 100644 project-personal/stage5/report/Makefile
 create mode 100644 project-personal/stage5/report/bib/cite.bib
 create mode 100644 project-personal/stage5/report/image/placeimg_800_600_tech.jpg
 create mode 100644 project-personal/stage5/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
 create mode 100755 project-personal/stage5/report/pandoc/filters/pandoc_eqnos.py
 create mode 100755 project-personal/stage5/report/pandoc/filters/pandoc_fignos.py
 create mode 100755 project-personal/stage5/report/pandoc/filters/pandoc_secnos.py
 create mode 100755 project-personal/stage5/report/pandoc/filters/pandoc_tablenos.py
 create mode 100644 project-personal/stage5/report/pandoc/filters/pandocxnos/__init__.py
 create mode 100644 project-personal/stage5/report/pandoc/filters/pandocxnos/core.py
 create mode 100644 project-personal/stage5/report/pandoc/filters/pandocxnos/main.py
 create mode 100644 project-personal/stage5/report/pandoc/filters/pandocxnos/pandocattributes.py
 create mode 100644 project-personal/stage5/report/report.md
 create mode 100644 project-personal/stage6/presentation/Makefile
 create mode 100644 project-personal/stage6/presentation/image/kulyabov.jpg
 create mode 100644 project-personal/stage6/presentation/presentation.md
 create mode 100644 project-personal/stage6/report/Makefile
 create mode 100644 project-personal/stage6/report/bib/cite.bib
 create mode 100644 project-personal/stage6/report/image/placeimg_800_600_tech.jpg
 create mode 100644 project-personal/stage6/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
 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
marudenko@dk3n60 ~/work/study/2023-2024/Операционные системы/os-intro $ git push
Перечисление объектов: 40, готово.
Подсчет объектов: 100% (40/40), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (30/30), готово.
Запись объектов: 100% (38/38), 341.45 КиБ | 15.52 МиБ/с, готово.
Всего 38 (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:Andre5789/study_2023-2024_os-intro.git
   df547e7..d8a4e98  master -> master
marudenko@dk3n60 ~/work/study/2023-2024/Операционные системы/os-intro $
```

6 Выводы

Здесь кратко описываются итоги проделанной работы.