

Лабораторная работа № 2

Первоначальная настройка git

Андреева Яна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	14
5	Контрольные вопросы	21
6	Выводы	23
7	Список литературы	24

Список иллюстраций

Список таблиц

1 Цель работы

Изучить идеологию и применение средств контроля версий и 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

`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

19:57

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 соз

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 core.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
```

4 Выполнение лабораторной работы

Установка программного обеспечения Установим git: `dnf install git` и установим

```
yaandreeva@yaandreeva:~$ sudo -i
[sudo] пароль для yaandreeva:
root@yaandreeva:~# dnf install git
Обновление и загрузка репозитория:
Fedora 42 - x86_64 - Updates
Fedora 42 - x86_64 - Updates
Репозитории загружены.
Пакет "git-2.49.8-1.fc42.x86_64" уже установлен.

Нечего делать.
root@yaandreeva:~# dnf install gh
Обновление и загрузка репозитория:
Репозитории загружены.
Пакет                                Арх.      Версия                                Репозиторий
Установка:
gh                                   x86_64    2.72.8-1.fc42                        updates
Сводка транзакции:
Установка:      1 пакета
Общий размер входящих пакетов составляет 11 MiB. Необходимо загрузить 11 MiB.
После этой операции будут использоваться дополнительные 39 MiB (установка 39 MiB, удаление 0 B).
Is this ok [y/N]:
```

gh dnf install gh

Базовая настройка 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` Зададим имя начальной ветки (будем называть её master): `git config --global init.defaultBranch master` Параметр autocrlf: `git config --global core.autocrlf input` Параметр safecrlf: `git config --global core.safecrlf warn`

```
root@yaandreeva:~# git config --global user.name "Yana Andreeva"
root@yaandreeva:~# git config --global user.email "yanandreecal@mail.ru"
root@yaandreeva:~# git config --global core.quotepath false
root@yaandreeva:~# git config --global init.defaultBranch mas
root@yaandreeva:~# git config --global core.autocrlf input
root@yaandreeva:~# git config --global core.safecrlf warn
root@yaandreeva:~#
```

Создадим ключи ssh по алгоритму rsa с ключём размером 4096 бит:
`ssh-keygen -t rsa -b 4096` по алгоритму ed25519: `ssh-keygen -t ed25519`

```

root@yaandreeva:~# ssh-keygen -t rsa -b 4096
-bash: ssh-keygen: команда не найдена
root@yaandreeva:~# ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase for "/root/.ssh/id_rsa" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:gWcks16QEr2zwPFJ7gMs+uHgbsr3CE+kf8WM2UX4fPQ root@yaandreeva.yaandreeva.net
The key's randomart image is:
+---[RSA 4096]-----+
|      .o+.o      |
|      o +0 . .   |
|      o *oo0 . .  |
|      . =.*+ = . E |
|      . o +BoS .   |
|      . = o+=      |
|      . = = ..     |
|      . 0.o.       |
|      . =o+o.      |
+---[SHA256]-----+
root@yaandreeva:~# ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
Enter passphrase for "/root/.ssh/id_ed25519" (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:fN3F9ro9S20G6vjLdu41l0h8wE/ymmf2IXCuzD+93Co root@yaandreeva.yaandreeva.net
The key's randomart image is:
+---[ED25519 256]---+
|      o . . |
|      = + |
|      . . . o . o |
|      S . . . . o . o |
|      = + o . o = + |
|      . 0o + + + o |
|      o . o X E . + * |
|      = 0 + * = 0 o |
+---[SHA256]-----+
root@yaandreeva:~#

```

Создадим ключи pgp Генерируем ключ gpg `--full-generate-key` Из предложенных опций выбираем: тип RSA and RSA; размер 4096; выберем 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 элек-

```

root@yaandreeva:~# gpg --full-generate-key
gpg (GnuPG) 2.4.7; Copyright (C) 2024 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

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) 0
Срок действия ключа не ограничен
Все верно? (y/N) y

GnuPG должен составить идентификатор пользователя для идентификации ключа.

Ваше полное имя: Yana Andreeva
Адрес электронной почты: yanandreeva@mail.ru
Примечание:
Вы выбрали следующий идентификатор пользователя:
  "Yana Andreeva <yanandreeva@mail.ru>"

Сменить (N)Имя, (C)Примечание, (E)Адрес; (O)Принять/(Q)Выход?

```

тронной почты.

Настройка github Создадим учётную запись на <https://github.com> и заполним основные данные.

Добавление PGP ключа в GitHub Выводим список ключей и копируем отпечаток приватного ключа: `gpg --list-secret-keys --keyid-format LONG`


```

root@yaandreeva:~# 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, 0m, 0f, 1u
[keyboard]
-----
sec   rsa4096/62F87252F8AEE5B1 2025-05-26 [SC]
      859F792A6537D23F96CEF49762F87252F8AEE5B1
uid           [ абсолютно ] Yana Andreeva <yanandreca1@mail.ru>
ssb   rsa4096/5F1C726F92BDFC8C 2025-05-26 [E]
root@yaandreeva:~#

```

Скопируем сгенерированный PGP ключ в буфер обмена: `gpg --armor --export | xclip -sel clip` Перейдем в настройки GitHub (<https://github.com/settings/keys>), нажмем на кнопку New GPG key и вставим полученный ключ в поле ввода.

Добавить новый ключ GPG

Название

Ключиша

```

-----BEGIN PGP PUBLIC KEY BLOCK-----
mQINBGlzcWUwOGRKpM4T0nqcmIiKDKZWw0/pTFGNZwO0DKS0jTtOC0taWCF3mU/XkaupvZ0F0
RiwcS8QzCtzY+IWKREoFga4BoEmmc7SjbL9FlaL/xblgqAs6mR65ZNOgjKxc1ek2
/aRa9c1oY8zMTRGIFPB1zveDMdRTcni9aIMDSMg6luh67t/NvLY0X3QKMP7blirR
6txT1IUkLAyEsoQJ0Nw6RZQ1eiOtAzve9bX3zpJdm1FGn/1ny9Vad7jmPS8qUi5
l4K7VAny94s/I2l4uHDd8mfOskN2VpsDb1bbt9xUtwZrENmRecIFNaL4Ct1bIWZG
ihZ3+P3gv8oQjxRxTOKKYhh32r1iubomonmEMUh0yXK2IMlzXZ6a6iwwfwqXw1ZFX
lsI=
=TSQI
-----END PGP PUBLIC KEY BLOCK-----

```

Добавить ключ GPG

Настройка автоматических подписей коммитов git Используя введенный email, укажем Git применять его при подписи коммитов: `git config --global user.signingkey`
`git config --global commit.gpgsign true` `git config --global gpg.program $(which gpg2)`

```

root@yaandreeva:~# git config --global user.signingkey 62F07252F0AEE5B1
root@yaandreeva:~# git config --global commit.gpgsign true
root@yaandreeva:~# git config --global gpg.program $(which gpg2)
root@yaandreeva:~#

```

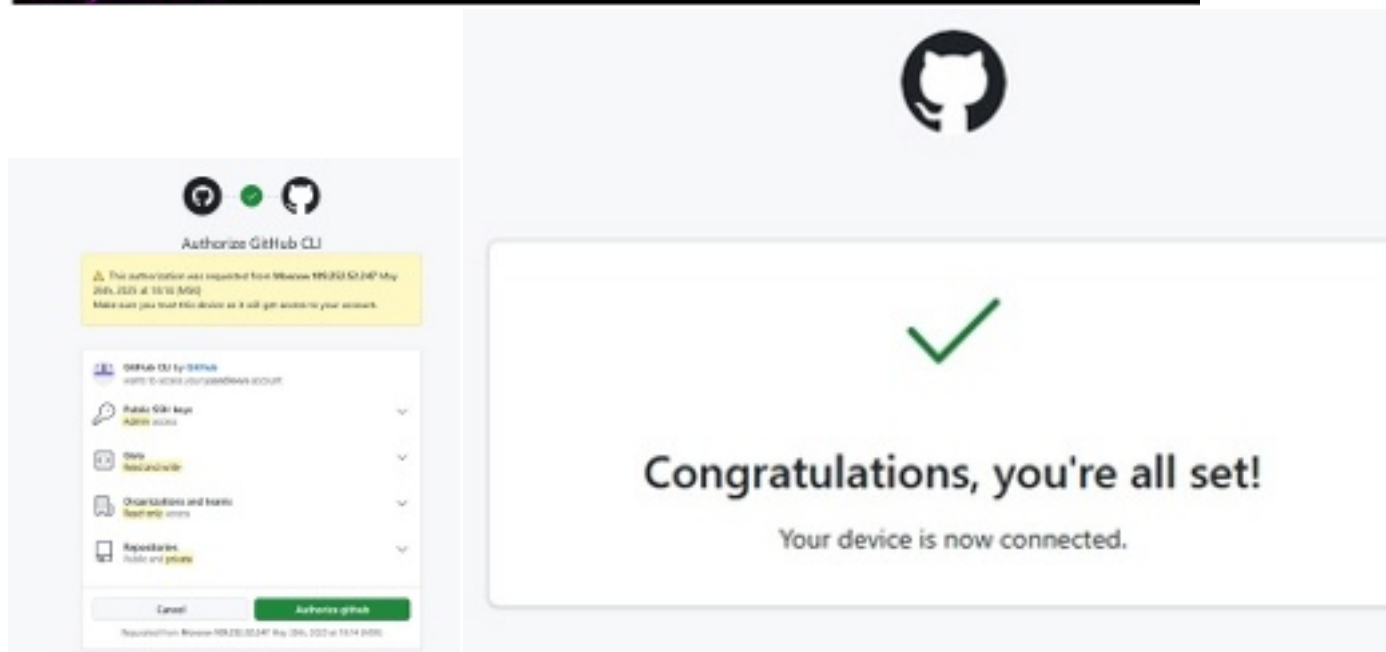
Настройка gh Для начала необходимо авторизоваться gh auth login Утилита задаст несколько наводящих вопросов. Авторизоваться можно через браузер.

```

root@yaandreeva:~# gh auth login
? Where do you use GitHub? GitHub.com
? What is your preferred protocol for Git operations on this host? SSH
? Upload your SSH public key to your GitHub account? /root/.ssh/id_ed25519.pub
? Title for your SSH key: GitHub CLI
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: A77C-E44D
Press Enter to open https://github.com/login/device in your browser...
/usr/sbin/xdg-open: строка 1045: www-browser: команда не найдена
/usr/sbin/xdg-open: строка 1045: links2: команда не найдена
/usr/sbin/xdg-open: строка 1045: elinks: команда не найдена
/usr/sbin/xdg-open: строка 1045: links: команда не найдена
/usr/sbin/xdg-open: строка 1045: lynx: команда не найдена
/usr/sbin/xdg-open: строка 1045: w3m: команда не найдена
xdg-open: no method available for opening 'https://github.com/login/device'
! Failed opening a web browser at https://github.com/login/device
exit status 3
Please try entering the URL in your browser manually
Authentication complete.
- gh config set -h github.com git_protocol ssh
Configured git protocol
! Authentication credentials saved in plain text
Uploaded the SSH key to your GitHub account: /root/.ssh/id_ed25519.pub
Logged in as yaandreeva
root@yaandreeva:~#

```



Шаблон для рабочего пространства Создадим репозиторий курса на основе шаблона `mkdir -p ~/work/study/2022-2023/"Операционные системы"` `cd ~/work/study/2022-2023/"Операционные системы"` `gh repo create study_2022-2023_os-intro --template=yamadharm/course-directory-student-template --public` `git clone --recursive git@github.com:/study_2022-2023_os-intro.git os-intro`

```
root@yaandreeva:~# mkdir -p ~/work/study/2024-2025/"Операционные системы"
root@yaandreeva:~# cd ~/work/study/2024-2025/"Операционные системы"
root@yaandreeva:~/work/study/2024-2025/Операционные системы# gh repo create study_2024-2025_os-intro --template=yamadharm/course-directory-student-template --public
Created repository yaandreeva/study_2024-2025_os-intro on github.com
https://github.com/yaandreeva/study_2024-2025_os-intro
root@yaandreeva:~/work/study/2024-2025/Операционные системы# git clone --recursive git@github.com:yaandreeva/study_2024-2025_os-intro.git os-intro
Клонирование в «os-intro»...
remote: Enumerating objects: 36, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (35/35), done.
remote: Total 36 (delta 1), reused 21 (delta 0), pack-reused 0 (from 0)
Получение объектов: 100% (36/36), 19.43 Кб | 504.00 Кб/с, готово.
Определение изменений: 100% (1/1), готово.
Подмодуль «template/presentation» (https://github.com/yamadharm/academic-presentation-markdown-template.git) зарегистрирован по пути «template/presentation»
Клонирование в «/root/work/study/2024-2025/Операционные системы/os-intro/template/presentation»...
remote: Enumerating objects: 147, done.
remote: Counting objects: 100% (147/147), done.
remote: Compressing objects: 100% (100/100), done.
remote: Total 147 (delta 55), reused 131 (delta 39), pack-reused 0 (from 0)
Получение объектов: 100% (147/147), 2.64 Мб | 4.17 Мб/с, готово.
Определение изменений: 100% (55/55), готово.
Клонирование в «/root/work/study/2024-2025/Операционные системы/os-intro/template/report»...
remote: Enumerating objects: 195, done.
remote: Counting objects: 100% (195/195), done.
remote: Compressing objects: 100% (133/133), done.
remote: Total 195 (delta 86), reused 161 (delta 52), pack-reused 0 (from 0)
Получение объектов: 100% (195/195), 766.76 Кб | 2.99 Мб/с, готово.
Определение изменений: 100% (86/86), готово.
Submodule path 'template/presentation': checked out '645759e4b104e93753637dedf8109adf24d071b7'
Submodule path 'template/report': checked out 'b5a97ed1ef36bf462109b7403402339ccaf27ea6'
root@yaandreeva:~/work/study/2024-2025/Операционные системы#
```

Настроим каталог курса

Перейдем в каталог курса:

`cd ~/work/study/2022-2023/"Операционные системы"/os-intro`

Удалим лишние файлы:

`rm package.json`

Создадим необходимые каталоги:

`echo os-intro > COURSE`

`make`

```
root@yaandreeva:~/work/study/2024-2025/Операционные системы# cd ~/work/study/2024-2025/"Операционные системы"/os-intro
root@yaandreeva:~/work/study/2024-2025/Операционные системы/os-intro# rm package.json
rm: удалить regular file 'package.json'? y
root@yaandreeva:~/work/study/2024-2025/Операционные системы/os-intro# rm package.json
rm: невозможно удалить 'package.json': Нет такого файла или каталога
root@yaandreeva:~/work/study/2024-2025/Операционные системы/os-intro# echo os-intro > COURSE
root@yaandreeva:~/work/study/2024-2025/Операционные системы/os-intro# make
Usage:
  make <target>

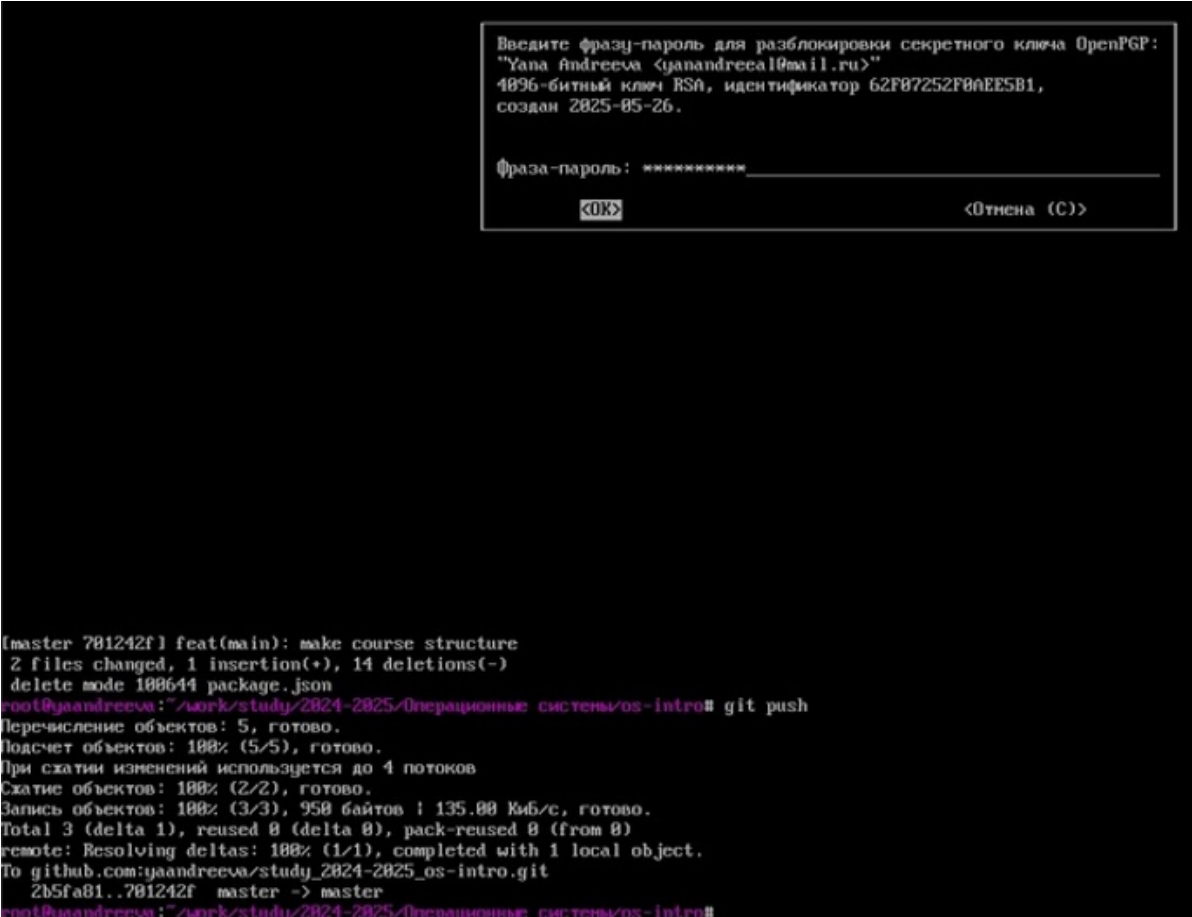
Targets:
  list           List of courses
  prepare        Generate directories structure
  submodule      Update submiles
root@yaandreeva:~/work/study/2024-2025/Операционные системы/os-intro#
```

Отправим файлы на сервер:

```
git add .
```

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

```
git push
```



```
Введите фразу-пароль для разблокировки секретного ключа OpenPGP:  
"Yana Andreeva <yaandreeva@mail.ru>"  
4096-битный ключ RSA, идентификатор 62F07252F80EE5B1,  
создан 2025-05-26.  
  
Фраза-пароль: *****  
[ОК] <Отмена (C)>
```

```
(master 701242f) feat(main): make course structure  
2 files changed, 1 insertion(+), 14 deletions(-)  
delete mode 100644 package.json  
root@yaandreeva:~/work/study/2024-2025/Операционные системы/os-intro# git push  
Перечисление объектов: 5, готово.  
Подсчет объектов: 100% (5/5), готово.  
При сжатии изменений используется до 4 потоков  
Сжатие объектов: 100% (2/2), готово.  
Запись объектов: 100% (3/3), 950 байтов | 135.00 Киб/с, готово.  
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)  
remote: Resolving deltas: 100% (1/1), completed with 1 local object.  
To github.com:yaandreeva/study_2024-2025_os-intro.git  
2b5fa81..701242f master -> master  
root@yaandreeva:~/work/study/2024-2025/Операционные системы/os-intro#
```

5 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? Системы контроля версий (Version Control Systems) — это инструменты для: -отслеживания изменений в файлах (чаще всего — в программном коде), -хранения истории изменений, -совместной работы над проектами, -возможности отката к предыдущим версиям.
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище (repository) — база данных всех изменений проекта. Commit — сохранённое состояние проекта с комментариями, метками времени и автором. История (history) — последовательность commit'ов. Рабочая копия (working copy) — текущие файлы проекта, с которыми работает пользователь.
3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. Централизованные VCS — один центральный сервер хранит всё (пример: Subversion/SVN). Плюсы: всё централизовано. Минусы: нельзя работать без подключения. Децентрализованные VCS — каждый участник имеет полную копию репозитория (пример: Git). Плюсы: автономность, скорость. Минусы: немного сложнее в координации.
4. Опишите действия с VCS при единоличной работе с хранилищем. Инициализация репозитория: `git init` Добавление файлов: `git add` Сохранение изменений: `git commit` Просмотр истории: `git log` Работа с ветками при необходимости: `git branch`, `git checkout`

5. Опишите порядок работы с общим хранилищем VCS. Клонирование: `git clone` Обновление локальной копии: `git pull` Отправка изменений: `git push` Разрешение конфликтов при слиянии изменений. Использование веток для параллельной работы: `git branch`, `git merge`
6. Каковы основные задачи, решаемые инструментальным средством git? Хранение истории изменений Ветвление и объединение изменений Работа офлайн Совместная работа Безопасное сохранение и откат версий
7. Назовите и дайте краткую характеристику командам git. `git init` — инициализация репозитория `git clone` — клонирование репозитория `git add` — добавление файлов к коммиту `git commit` — сохранение изменений `git status` — текущее состояние `git log` — история `git branch` — работа с ветками `git merge` — слияние веток `git pull` — получение и объединение изменений `git push` — отправка изменений
8. Приведите примеры использования при работе с локальным и удалённым репозиториями. Локальный: `git init` `git add .` `git commit -m "Первый коммит"` Удалённый: `git clone git@github.com:user/project.git` `git push origin main` `git pull origin main`
9. Что такое и зачем могут быть нужны ветви (branches)? Ветви — параллельные версии проекта, которые позволяют: -работать над новой функциональностью, -исправлять ошибки, не мешая основной версии (main/master), -тестировать. Примеры: `git branch feature-x` `git checkout feature-x`
10. Как и зачем можно игнорировать некоторые файлы при commit? Файлы, которые не нужно отслеживать (например, временные, конфигурации), можно указать в `.gitignore`. Пример `.gitignore`: `.log` `.tmp` `.env` `node_modules/`

6 Выводы

Изучили идеологию и применение средств контроля версий и освоили умения по работе с git.

7 Список литературы