

# **Отчёт по лабораторной работе №2**

**дисциплина: Операционные системы**

**BUTERIN ARSENIY**

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Контрольные вопросы</b>	<b>14</b>
<b>5</b>	<b>Выводы</b>	<b>17</b>

## Список иллюстраций

3.1	Базовая настройка git . . . . .	8
3.2	Создание ключа SSH . . . . .	9
3.3	Ключ SSH создан . . . . .	9
3.4	Ключ GPG создан . . . . .	9
3.5	Отпечаток приватного ключа . . . . .	10
3.6	Настройка подписей . . . . .	10
3.7	Создание репозитория . . . . .	11
3.8	Настраиваем каталог курса . . . . .	12

## 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 (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. # Выполнение лабораторной работы

Базовая настройка git:

1. Задаём имя и email владельца репозитория (1 и 2 строка на рисунке)
2. Настраиваем utf-8 в выводе сообщений git (3 строка на рисунке)
3. Настраиваем верификацию и подписание коммитов git. Зададим имя начальной ветки (будем называть её master) (4 строка на рисунке)

```
agbuterin@dk3n31 ~$ git config --global user.name "AGButerin"
agbuterin@dk3n31 ~$ git config --global user.email "senyabankai@gmail.com"
agbuterin@dk3n31 ~$ git config --global core.quotepath false
agbuterin@dk3n31 ~$ git config --global init.defaultBranch master
agbuterin@dk3n31 ~$ git config --global core.autocrlf input
agbuterin@dk3n31 ~$ git config --global core.safecrlf warn
agbuterin@dk3n31 ~$
```

Рис. 3.1: Базовая настройка git



Создаём ключ SSH. В терминале вводим данную команду:

```
ssh-keygen -t rsa -b 4096
```

Далее во всех пунктах пользуемся клавишей Enter и получаем наш ключ.

```
agbuterin@dk3n31 ~$ gpg --full-generate-key
gpg (GnuPG) 2.2.42; Copyright (C) 2023 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 и RSA (по умолчанию)
```

Рис. 3.2: Создание ключа SSH

Ключ нужно добавить на github. Для этого переходим на сайте в раздел “Settings” и выбираем “SSH and GPG keys”.

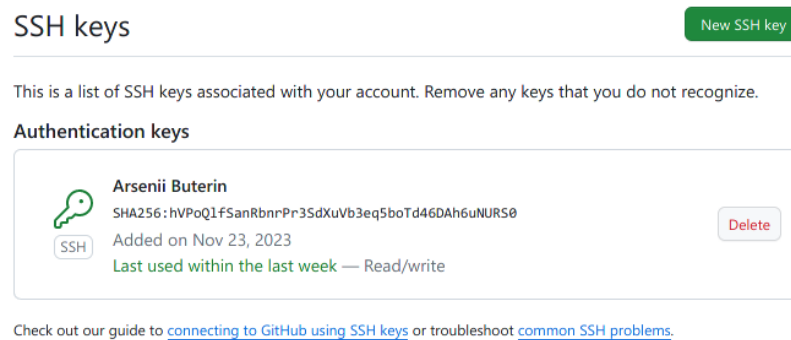


Рис. 3.3: Ключ SSH создан

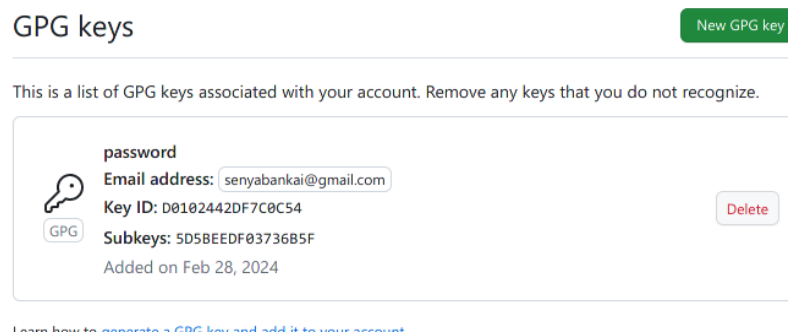


Рис. 3.4: Ключ GPG создан

Выводим список ключей и копируем отпечаток приватного ключа

```

agbuterin@dk3n31 ~ $ gpg --full-generate-key
gpg (GnuPG) 2.2.42; Copyright (C) 2023 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 и RSA (по умолчанию)
  (2) DSA и Elgamal
  (3) DSA (только для подписи)
  (4) RSA (только для подписи)
  (14) Имеющийся на карте ключ
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? (у/Н) у

GnuPG должен составить идентификатор пользователя для идентификации ключа.

Ваше полное имя: AGButerin
Адрес электронной почты: senyabankai@gmail.com
Примечание:
Вы выбрали следующий идентификатор пользователя:
  УПО: senyabankai@gmail.com

```

Рис. 3.5: Отпечаток приватного ключа

## Настройка автоматических подписей коммитов git

```

agbuterin@dk3n31 ~ $ git config --global user.signingkey <PGP Fingerprint>
bash: синтаксическая ошибка рядом с неожиданным маркером «newline»
agbuterin@dk3n31 ~ $ git config --global user.signingkey D0102442DF7C0C54
agbuterin@dk3n31 ~ $ git config --global commit.gpgsign true
agbuterin@dk3n31 ~ $ git config --global gpg.program $(which gpg2)
agbuterin@dk3n31 ~ $ gh auth login
? What account do you want to log into? GitHub.com

```

Рис. 3.6: Настройка подписей

Возвращаемся в наш терминал и настраиваем gh командой:

gh auth login.

Во всех пунктах выбираем у(yes).

По полученной ссылке переходим в браузер на виртуальной машине и вводим код из терминала (находится перед ссылкой).

```

Submodule path 'template/report': checked out '7c31ab8e591a8cd02d07caeb8a19e18028ced88e'
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы $ ls
study_2022-2023_os-intro
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы $ cd study_2022-2023_os-intro
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $ rm package.json
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $ echo os-intro > COURSE

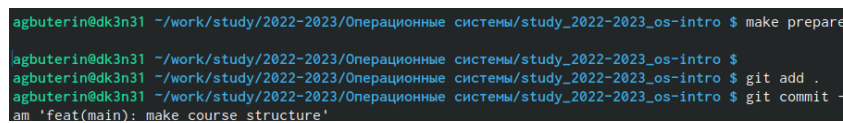
```

#fig:007

width=70% }

Создаём репозиторий курса на основе шаблона. Все нужные команды для создания были в указаниях к лабораторной работе. В 4 команде, вместо , указываем своё имя профиля на github.

1. `mkdir -p ~/work/study/2021-2022/“Операционные системы”`
2. `cd ~/work/study/2021-2022/“Операционные системы”`
3. `gh repo create study_2021-2022_os-intro --template=yamadharm/course-directory-student-template --public`
4. `git clone --recursive git@github.com:/study_2021-2022_os-intro.git os-intro`

A screenshot of a terminal window showing a series of commands and their outputs. The user is in a directory named 'study\_2022-2023\_Операционные системы/study\_2022-2023\_os-intro'. They run 'make prepare', then 'git add .', and finally 'git commit -am 'feat(main): make course structure''. The terminal output shows the commit message and the creation of the repository.

```
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $ make prepare
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $ git add .
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $ git commit -am 'feat(main): make course structure'
```

Рис. 3.7: Создание репозитория

Настраиваем каталог курса. Для этого переходим в него командой:

```
cd ~/work/study/2021-2022/“Операционные системы”/os-intro
```

Далее командой `ls` проверяем, что мы в него перешли. В каталоге “os-intro” нам потребуется удалить файл “package.json”. Выполняем данную задачу командой:

```
rm package.json
```

Снова командой `ls` проверяем успешное выполнение удаления файла.

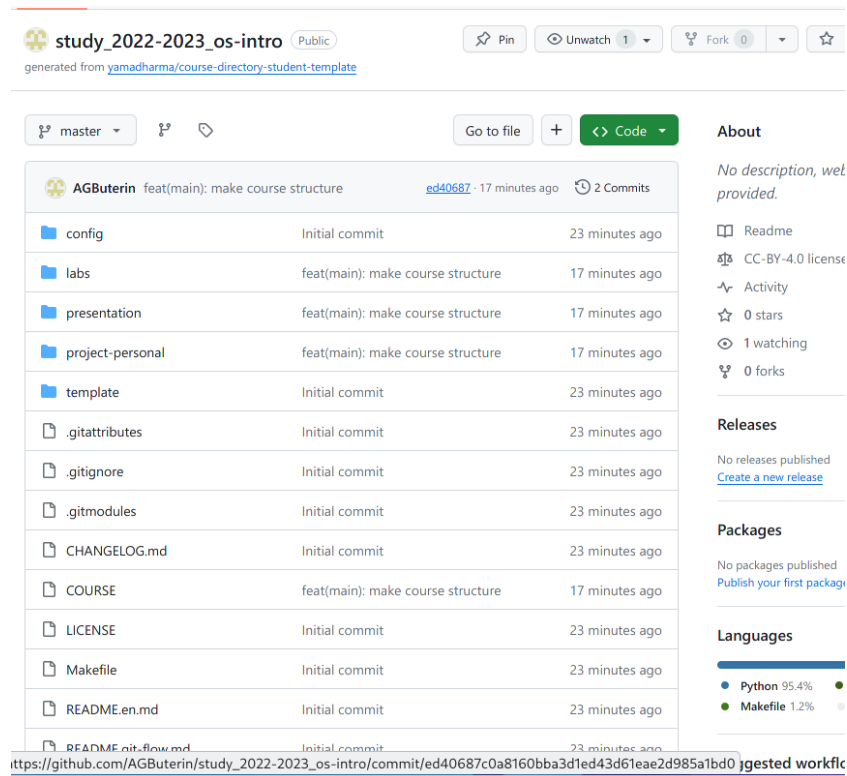


Рис. 3.8: Настраиваем каталог курса

Создаём необходимые каталоги и отправляем наши файлы на сервер  
make COURSE=os-intro

1. git add .
2. git commit -am 'feat(main): make course structure'
3. git push

```

agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $
am 'feat(main): make course structure'
error: gpg failed to sign the data:
[GNUPG:] KEY_CONSIDERED 4ED054CEB536FAE469AEC98DD0102442DF7C0C54 2
[GNUPG:] BEGIN_SIGNING H8
[GNUPG:] PINENTRY_LAUNCHED 4992 gnome3 1.2.1-unknown - xterm-256color :1 - 5429/10000 0
gpg: подписать не удалось: Операция отменена
[GNUPG:] FAILURE sign 83886179
gpg: signing failed: Операция отменена

fatal: сбой записи объекта коммита
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $
am 'feat(main): make course structure'
[master ed40687] feat(main): make course structure
361 files changed, 98413 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
create mode 100755 labs/lab01/report/pandoc/filters/pandoc_secnos.py
create mode 100755 labs/lab01/report/pandoc/filters/pandoc_tablenos.py
create mode 100644 labs/lab01/report/pandoc/filters/pandocxnos/__init__.py
create mode 100644 labs/lab01/report/pandoc/filters/pandocxnos/core.py
create mode 100644 labs/lab01/report/pandoc/filters/pandocxnos/main.py
create mode 100644 labs/lab01/report/pandoc/filters/pandocxnos/pandocattributes.py
create mode 100644 labs/lab01/report/report.md
create mode 100644 labs/lab02/presentation/Makefile
create mode 100644 labs/lab02/presentation/image/kulyabov.jpg

```

```

create mode 100644 project-personal/stage6/report/report.md
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/st
Перечисление объектов: 40, готово.
Подсчет объектов: 100% (40/40), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (30/30), готово.
Запись объектов: 100% (38/38), 342.10 КиБ | 2.65 МБ/с, готово.
Всего 38 (изменений 4), повторно использовано 0 (изменений 0),
remote: Resolving deltas: 100% (4/4), completed with 1 local ob
To github.com:AGButerin/study_2022-2023_os-intro.git
   3583624..ed40687  master -> master
agbuterin@dk3n31 ~/work/study/2022-2023/Операционные системы/st

```

## 4 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? Это программное обеспечение для облегчения работы с изменяющейся информацией. VCS 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.
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище (repository), или репозиторий, — место хранения всех версий и служебной информации. Commit («[трудовой] вклад», не переводится) — синоним версии; процесс создания новой версии. 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 VCS: одно основное хранилище всего проекта и каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет и, затем, добавляет свои изменения обратно. Децентрализованные VCS: у каждого пользователя свой вариант (возможно не один) репозитория.
4. Опишите действия с VCS при единоличной работе с хранилищем.
5. Опишите порядок работы с общим хранилищем VCS.

6. Каковы основные задачи, решаемые инструментальным средством git? Git — это система управления версиями. У Git две основных задачи: первая — хранить информацию о 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. git –version (Проверка версии Git) git init (Инициализировать ваш текущий рабочий каталог как Git-репозиторий) git clone <https://www.github.com/username/repo-name> (Скопировать существующий удаленный Git-репозиторий) git remote (Просмотреть список текущих удалённых репозиториях Git) git remote -v (Для более подробного вывода) git add my\_script.py (Можете указать в команде конкретный файл). git add . (Позволяет охватить все файлы в текущем каталоге, включая файлы, чье имя начинается с точки) git commit -am “Commit message” (Вы можете сжать все индексированные файлы и отправить коммит). git branch (Просмотреть список текущих веток можно с помощью команды branch) git –help (Чтобы узнать больше обо всех доступных параметрах и командах) git push origin master (Передать локальные коммиты в ветку удаленного репозитория).
8. Приведите примеры использования при работе с локальным и удалённым репозиториями.
9. Что такое и зачем могут быть нужны ветки (branches)? 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.



## **5 Выводы**

В ходе выполнения лабораторной работы изучили идеологию и применение средств контроля версий, а также освоили умения по работе с git.