

# Лабораторная работа №3

Курс “Операционные Системы”

---

Гибшер К.В. , НКАбд-01-22

23 февраля 2023

Российский университет дружбы народов, Москва, Россия

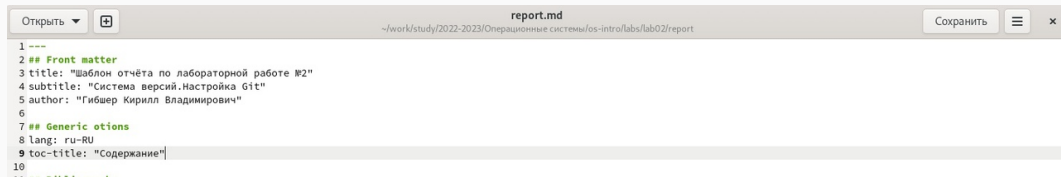
- Гибшер Кирилл Владимирович
- студент группы НКАбд-01-22
- кафедры Компьютерные и информационные науки
- Российский университет дружбы народов
- kirill.gibsher@gmail.com

- Научиться оформлять отчёты с помощью легковесного языка разметки Markdown.
1. Сделать отчет по предыдущей лабораторной работе в Markdown
  2. Предоставить отчет в форматах doc,pdf,md.

## Выполнение лабораторной работы

---

# Создание шапки отчета



The screenshot shows a code editor window titled "report.md" with a file path of "~/work/study/2022-2023/Операционные системы/os-intro/labs/lab02/report". The editor contains the following Markdown code:

```
1 ---
2 ## Front matter
3 title: "Шаблон отчёта по лабораторной работе №2"
4 subtitle: "Система версий.Настройка Git"
5 author: "Гибшер Кирилл Владимирович"
6
7 ## Generic otions
8 lang: ru-RU
9 toc-title: "Содержание"
10
11 -- Additional content --
```

The interface includes a toolbar with "Открыть" (Open) and a "+" icon, and a right sidebar with "Сохранить" (Save), a menu icon, and a close icon "x".

Рис. 1: Создание шапки отчета

# Демонстрация целей и задач



```
60 %> \usepackage{listings}
61 ## Misc options
62 indent: true
63 header-includes:
64   - \usepackage{indentfirst}
65   - \usepackage{float} # keep figures where there are in the text
66   - \floatplacement{figure}{H} # keep figures where there are in the text
67 ---
68
69
70 # Цель работы
71
72 - Изучить идеологию и применение средств контроля версий.
73 - Освоить умения по работе с git.
74
75 # Задание
76
77 1. Создать базовую конфигурацию для работы с git.
78
79 2. Создать ключ SSH.
80
81 3. Создать ключ PGP.
82
83 4. Настроить подписи git.
84
85 5. Зарегистрироваться на Github.
86
87 6. Создать локальный каталог для выполнения заданий по предмету.
88
```

Рис. 2: Цели и задачи

# Теоретическое введение лабораторной работы

## 1 часть теор. части.

```
88
89 # Теоретическое введение
90
91 ## Системы контроля версий. Общие понятия
92
93 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 полные версии изменённых файлов, а производить так называемую дельта-компрессию – сохранять только изменения между последовательными версиями, что позволяет
уменьшать 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 изменений. 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 ветки. 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.
94
95 ## Основные команды git
96
97 Перечислим наиболее часто используемые команды git.
98
99 - Создание основного дерева репозитория: git init
100
101 - Получение обновлений (изменений) текущего дерева из центрального репозитория: git pull
102
103 - Отправка всех произведённых изменений локального дерева в центральный репозиторий: git push
104
105 - Просмотр списка изменённых файлов в текущей директории: git status
106
107 - Просмотр текущих изменений: git diff
108
109 - добавить все изменённые и/или созданные файлы и/или каталоги: git add .
110
111 - добавить конкретные изменённые и/или созданные файлы и/или каталоги: git add имена_файлов
112
113 - удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm имена_файлов
114
115 - сохранить все добавленные изменения и все изменённые файлы: git commit -am 'Описание коммита'
116
117 - сохранить добавленные изменения с внесением комментария через встроенный редактор: git commit
118
119 - создание новой ветки, базирующейся на текущей: git checkout -b имя_ветки
120
121 - 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)
122
123 - отправка изменений конкретной ветки в центральный репозиторий: git push origin имя_ветки
124
125 - слияние ветки с текущим деревом: git merge --no-ff имя_ветки
126
127 - удаление локальной уже слитой с основным деревом ветки: git branch -d имя_ветки
128
129 - принудительное удаление локальной ветки: git branch -D имя_ветки
130
131 - удаление ветки с центрального репозитория: git push origin :имя_ветки
```

# Теоретическое введение лабораторной работы

## 2 часть теор. части.

```
report.md
~www.kitstudy/2022-2023/Операционные системы/ос-интро/файл02/report

206 - Шифры 3des-cbc и arcfour потенциально уязвимы в силу использования DES и RC4.
207
208 - Шифр cast128-cbc применяет слишком короткий размер блока (64 бит).
209
210 3. Обмен ключами
211
212 - Применяемые в SSH методы обмена ключей DH (Diffie-Hellman) и ECDH (Elliptic Curve Diffie-Hellman) можно считать безопасными.
213
214 - Из 8 поддерживаемых в SSH протоколах обмена ключами вызывает подозрения три, основанные на рекомендациях NIST:
215
216 - ecdh-sha2-nistp256;
217
218 - ecdh-sha2-nistp384;
219
220 - ecdh-sha2-nistp521.
221
222 - Не стоит использовать протоколы, основанные на SHA1.
223
224 4. Файлы ssh-ключей
225
226 - По умолчанию пользовательские ssh-ключи сохраняются в каталоге ~/.ssh в домашнем каталоге пользователя.
227
228 - Убедитесь, что у вас ещё нет ключа.
229
230 - Файлы закрытых ключей имеют названия типа id_алгоритм (например, id_dsa, id_rsa).
231
232 - По умолчанию закрытые ключи имеют имена:
233
234 - id_dsa
235
236 - id_ecdsa
237
238 - id_ed25519
239
240 - id_rsa
241
242 - Открытые ключи имеют дополнительные расширения .pub.
243
244 - По умолчанию публичные ключи имеют имена:
245
246 - id_dsa.pub
247
248 - id_ecdsa.pub
249
250 - id_ed25519.pub
251
252 - id_rsa.pub
253
254 - При создании ключа команда попросит ввести любую ключевую фразу для более надёжной защиты вашего пароля. Можно пропустить этот этап, нажав Enter.
255
256 - Сменить пароль на ключ можно с помощью команды: ssh-keygen -p
257
258 ## Верификация коммитов с помощью PGP
259
260 Коммиты имеют следующие свойства:
261
262 - author (автор) – контрибьютор, выполнивший работу (указывается для справки); committer (коммитер) – пользователь, который закоммитил изменения.
263
264 - Эти свойства можно переопределить при совершении коммита.
265
266 - Авторство коммита можно подделывать.
```



# Запись хода лабораторной работы

```
report.md
~\work\study\2022-2023\Операционные системы\os-intro\labs\lab02\report

274 # Выполнение лабораторной работы
275
276 Создаю аккаунт на гитхаб. (рис. [-@fig:001]).
277
278 ![Создание GitHub](image/1.jpg){#fig:001 width=70%}
279
280 Так как git у меня уже был установлен, произвожу установку gh с помощью команды dnf install gh (рис. [-@fig:002]).
281
282 ![Установка gh](image/2.jpg){#fig:002 width=70%}
283
284 Затем приступаю к базовой настройке гит и прописываю некоторые параметры с помощью следующих команд: git config --global core.autocrlf input / git config --global
core.safecrlf warn (рис. [-@fig:003]).
285
286 ![Базовая настройка Git](image/3.jpg){#fig:003 width=70%}
287
288 Сгенерирую ключ rsa с размером 4096 с помощью команды ssh-keygen -t rsa -b 4096 и ключ по алгоритму ed25519 ssh-keygen -t ed25519 (рис. [-@fig:004]).
289
290 ![Генерация ключей SSH](image/4.jpg){#fig:004 width=70%}
291
292 Создадим PGP ключ с помощью команды gpg --full-generate-key со следующими параметрами: тип RSA и RSA; размер 4096; 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 (рис. [-@fig:005]).
293
294 ![Генерация PGP ключа](image/5.jpg){#fig:005 width=70%}
295
296
297 Далее необходимо скопировать сгенерированный ключ для того, чтобы сохранить его на гитхабе. Сделаем это с помощью команды. Выводим список ключей и копируем отпечаток
приватного ключа: gpg --list-secret-keys --keyid-format LONG \ gpg --armor --export PGP Fingerprint | xclip -sel clip. Перейдем в настройки GitHub, нажмем на кнопку New GPG
key и вставим полученный ключ в поле ввода. (рис. [-@fig:006]).
298
299 ![Добавление PGP ключа на гитхаб](image/6.jpg){#fig:006 width=70%}
300
301 Ключ успешно сформирован (рис. [-@fig:007]).
302
303 ![Ключ на GitHub](image/7.jpg){#fig:007 width=70%}
304
305 Далее необходимо настроить gh для этого авторизируемся с помощью команды gh auth (рис. [-@fig:008]).
306
307 ![Авторизация gh](image/8.jpg){#fig:008 width=70%}
308
309 Создадим рабочее пространство курса в нашей ОС и затем с помощью команды git clone скопируем материалы курса с github (рис. [-@fig:009]).
310
311 ![Настройка рабочего пространства](image/9.jpg){#fig:009 width=70%}
312
313 Далее приступим к настройке каталога курса, для этого удалим лишние файлы командой rm package.json. Создадим необходимые каталоги echo os-intro > COURSE и make. (рис. [-@fig:
010]).
314
315 ![Настройка каталогов курса](image/10.jpg){#fig:010 width=70%}
316
317 Последним шагом отправим файлы на сервер с помощью git add git commit и git push. (рис. [-@fig:011])
318
319 ![Отправка файлов на сервер](image/11.jpg){#fig:011 width=70%}
```

```
389
390
391 # Выводы
392
393 Таким образом, благодаря данной лабораторной работе я приобрел практических навыков установки операционной системы на виртуальную машину, настройки минимально необходимых для
    дальнейшей работы сервисов.
394
395
396 # Список литературы[.unnumbered]
397
398 1. О системе контроля версий [Электронный ресурс] - Режим доступа:https://git-scm.com/book/ru/v2/Введение-0-системе-контроля-версий
399
400 2. Системы контроля версий [Электронный ресурс] - Режим доступа: http://uii.mpei.ru/study/courses/sdt/16/lecture02.2_vcs.slides.pdf.
401
402
403
404
405 ::: {#refs}
406 :::
```

Парная скобка найдена в строке: 396

Markdown ▾    Ширина табуляции: 8 ▾    Стр 396, Стлб 33 ▾    ВСТ

Рис. 6: Выводы

## Результаты

---

Изучив способности Markdown мы может намного легче составлять отчеты по лабораторным работам!

...