

Отчёт по лабораторной работе №2

дисциплина: Операционные системы

Саакян Нерсес Варданович

Содержание

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

Список иллюстраций

3.1	Базовая настройка git	9
3.2	Создание ключа SSH	9
3.3	Создание ключа GPG	9
3.4	Ключ SSH создан	10
3.5	Ключ GPG создан	10
3.6	Отпечаток приватного ключа	10
3.7	Настройка подписей	11
3.8	Настройка gh	11
3.9	Создание репозитория	12
3.10	Настраиваем каталог курса	12
3.11	Отправляем наши файлы на сервер	13

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)
4. Параметр autocrlf (5 строка на рисунке)
5. Параметр safecrlf (6 строка на рисунке)


```
nvsaakyan@dk6n53 ~ $ git config --global user.name "saakyannerses"
nvsaakyan@dk6n53 ~ $ git config --global user.email "1132239125@pfur.ru"
nvsaakyan@dk6n53 ~ $ git config --global core.quotepath false
nvsaakyan@dk6n53 ~ $ git config --global init.defaultBranch master
nvsaakyan@dk6n53 ~ $ git config --global core.autocrlf input
nvsaakyan@dk6n53 ~ $ git config --global core.safecrlf warn
```

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.

```
nvsaakyan@dk6n53 ~ $ 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/n/v/nvsaakyan/.ssh/id_rsa):
/afs/.dk.sci.pfu.edu.ru/home/n/v/nvsaakyan/.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/n/v/nvsaakyan/.ssh/id_rsa
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/n/v/nvsaakyan/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:IMrCWcTZN7TMjFXVR70CBiFJFEQ2WxZBJBiskuXWA0 nvsaakyan@dk6n53
The key's randomart image is:
+---[RSA 4096]-----+
|  E.=@+o..o |
|..o..X.+...o|
|o..+..o...o|
|..o..+..o...|
|..o..+..o...|
|...+..S...|
|..o..|
|..o..|
|..|
|..|
+---[SHA256]-----+
```

Рис. 3.2: Создание ключа SSH

```
nvsaakyan@dk6n53 ~ $ 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/n/v/nvsaakyan/.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/n/v/nvsaakyan/.ssh/id_ed25519
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/n/v/nvsaakyan/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:xNziVYVSzA2cqXnrlvPezE5cjP22S9tFXRcW06wEVBs nvsaakyan@dk6n53
The key's randomart image is:
+--[ED25519 256]--+
| ..+ooE+o|
| o ++o oo=|
| =o= ....o|
| oo+.. .++|
| S.o.. . *|
| ..o . o.|
| ... o.+|
| = ..=|
| . +o.=+o|
+---[SHA256]-----+
```

Рис. 3.3: Создание ключа GPG

Ключ нужно добавить на github. Для этого переходим на сайте в раздел

“Settings” и выбираем “SSH and GPG keys”.

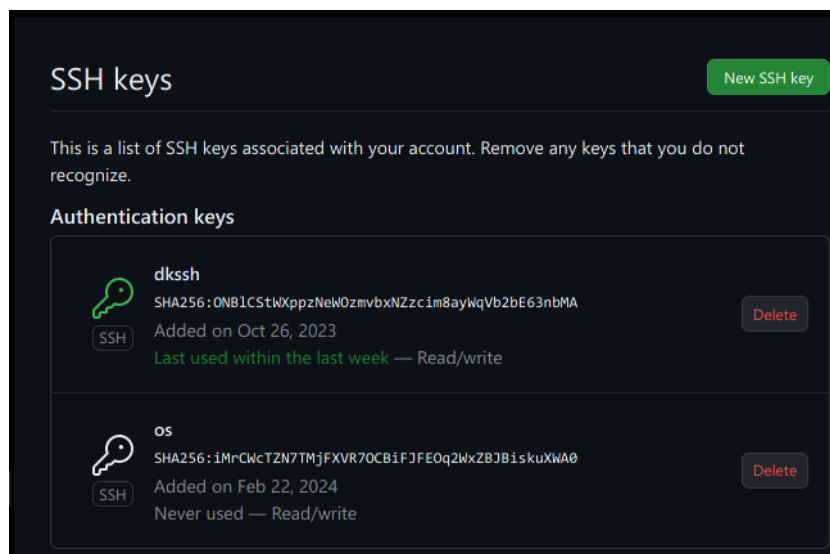


Рис. 3.4: Ключ SSH создан

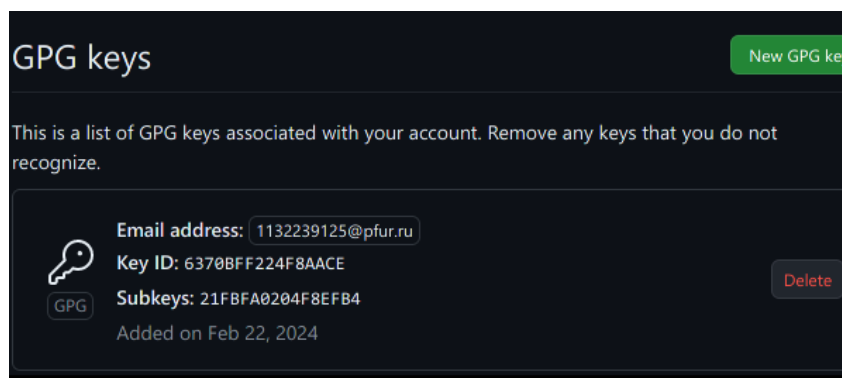


Рис. 3.5: Ключ GPG создан

Выводим список ключей и копируем отпечаток приватного ключа

```
nvsaakyan@dk6n53 ~ $ 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
/afs/.dk.sci.pfu.edu.ru/home/n/v/nvsaakyan/.gnupg/pubring.kbx
-----
sec   rsa4096/6370BFF224F8AAACE 2024-02-22 [SC]
      54C62A64FA5FAAF20F76ACDA6370BFF224F8AAACE
uid           [ абсолютно ] nvsaakyan <1132239125@pfur.ru>
ssb   rsa4096/21FBFA0204F8EFB4 2024-02-22 [E]
```

Рис. 3.6: Отпечаток приватного ключа

Настройка автоматических подписей коммитов git

```
nvsaaakyan@dk6n53 ~ $ git config --global user.signingkey 6370BFF224F8AAACE
nvsaaakyan@dk6n53 ~ $ git config --global commit.gpgsign true
nvsaaakyan@dk6n53 ~ $ git config --global gpg.program $(which gpg2)
```

Рис. 3.7: 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).

```
nvsaaakyan@dk6n53 ~ $ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations on this host? HTTPS
\
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser

  First copy your one-time code: 8F72-C1A8
Press Enter to open github.com in your browser...
[47276:47276:0222/170930.579362:ERROR:object_proxy.cc(576)] Failed to call method: org.freedesktop.ScreenSaver.GetActive:
object_path= /org/freedesktop/ScreenSaver: org.freedesktop.DBus.Error.NotSupported: This method is not part of the idle
inhibition specification: https://specifications.freedesktop.org/idle-inhibit-spec/latest/
[47276:47276:0222/170930.582005:ERROR:object_proxy.cc(576)] Failed to call method: org.gnome.ScreenSaver.GetActive: objec
t_path= /org/gnome/ScreenSaver: org.freedesktop.DBus.Error.ServiceUnknown: GDBus.Error:org.freedesktop.DBus.Error.Service
Unknown: The name org.gnome.Shell.ScreenShield was not provided by any .service files
[47276:47313:0222/170930.590000:ERROR:nss_util.cc(357)] After loading Root Certs, loaded==false: NSS error code: -8018
[47276:47276:0222/170939.153413:ERROR:download_item_impl.cc(2476)] Download full path should be empty before resumption
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
~ Authentication complete.
~ gh config set -h github.com git_protocol https
~ Configured git protocol

~ Logged in as saakyannerses
nvsaaakyan@dk6n53 ~ $
```

Рис. 3.8: Настройка gh

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=yamadharma/course-directory-student-template --public`

4. git clone –recursive git@github.com:/study_2021-2022_os-intro.git os-intro

```
nvsaakyan@dk6n53 ~$ mkdir -p ~/work/study/2023-2024/"Операционные системы"
nvsaakyan@dk6n53 ~$ cd ~/work/study/2023-2024/"Операционные системы"
nvsaakyan@dk6n53 ~/work/study/2023-2024/Операционные системы$ gh repo create study_2023-2024_os-intro --template=yamadharma/course-directory-student-template --public
✓ Created repository saakyannurses/study_2023-2024_os-intro on GitHub
nvsaakyan@dk6n53 ~/work/study/2023-2024/Операционные системы$ git clone --recursive git@github.com:saakyannurses/study_2023-2024_os-intro.git os-intro
Клонирование в «os-intro»...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 32 (delta 1), reused 18 (delta 0), pack-reused 0
Получение объектов: 100% (32/32), 18.60 Киб | 906.00 Киб/с, готово.
Определение изменений: 100% (1/1), готово.
Подмодуль «template/presentation» (https://github.com/yamadharma/academic-presentation-markdown-template.git) зарегистрирован по пути «template/presentation»
Подмодуль «template/report» (https://github.com/yamadharma/academic-laboratory-report-template.git) зарегистрирован по пути «template/report»
Клонирование в «/afs/.dk.sci.pfu.edu.ru/home/n/v/nvsaakyan/work/study/2023-2024/Операционные системы/os-intro/template/presentation»...
remote: Enumerating objects: 95, done.
remote: Counting objects: 100% (95/95), done.
remote: Compressing objects: 100% (67/67), done.
remote: Total 95 (delta 34), reused 87 (delta 26), pack-reused 0
Получение объектов: 100% (95/95), 96.99 Киб | 176.00 Киб/с, готово.
Определение изменений: 100% (34/34), готово.
Клонирование в «/afs/.dk.sci.pfu.edu.ru/home/n/v/nvsaakyan/work/study/2023-2024/Операционные системы/os-intro/template/report»...
remote: Enumerating objects: 126, done.
remote: Counting objects: 100% (126/126), done.
remote: Compressing objects: 100% (87/87), done.
```

Рис. 3.9: 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.

```
nvsaakyan@dk6n53 ~/work/study/2023-2024/Операционные системы$ cd ~/work/study/2023-2024/"Операционные системы"/os-intro
nvsaakyan@dk6n53 ~/work/study/2023-2024/Операционные системы/os-intro$ rm package.json
nvsaakyan@dk6n53 ~/work/study/2023-2024/Операционные системы/os-intro$ echo os-intro > COURSE
nvsaakyan@dk6n53 ~/work/study/2023-2024/Операционные системы/os-intro$ make
Usage:
  make <target>

Targets:
  list           List of courses
  prepare       Generate directories structure
  submodule     Update submodules
```

Рис. 3.10: 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`

```
nvsaakyan@dk6n53 ~/work/study/2023-2024/Операционные системы/os-intro $ git add .
nvsaakyan@dk6n53 ~/work/study/2023-2024/Операционные системы/os-intro $ git commit -am 'feat(main): make course structure'
[master 15feef3] feat(main): make course structure
 2 files changed, 1 insertion(+), 14 deletions(-)
 delete mode 100644 package.json
nvsaakyan@dk6n53 ~/work/study/2023-2024/Операционные системы/os-intro $ git pushПеречисление объектов: 5, готово.
Подсчет объектов: 100% (5/5), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (2/2), готово.
Запись объектов: 100% (3/3), 954 байта | 954.00 КиБ/с, готово.
Всего 3 (изменений 1), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:saakyannerses/study_2023-2024_os-intro.git
   4dcfc35..15feef3  master -> master
nvsaakyan@dk6n53 ~/work/study/2023-2024/Операционные системы/os-intro $
```

Рис. 3.11: Отправляем наши файлы на сервер

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.