

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

Дисциплина: Операционные системы

Студент: Дарижапов Тимур Андреевич

Группа: НПМбд-01-20

МОСКВА

2020 г.

Цель: изучить идеологию и применение средств контроля версий

Ход работы: 1. Настройка git

```
[tadarizhapov@tadarizhapov ~]$ git config --global user.name "Timur Darizhapov"
[tadarizhapov@tadarizhapov ~]$ git config --global user.email "timaanddar@mail.ru"
[tadarizhapov@tadarizhapov ~]$ git config --global quotepath false
```

рис.1

Создаём локальный репозиторий

```
[tadarizhapov@tadarizhapov ~]$ cd
[tadarizhapov@tadarizhapov ~]$ mkdir laboratory
[tadarizhapov@tadarizhapov ~]$ cd laboratory
[tadarizhapov@tadarizhapov laboratory]$ git init
Initialized empty Git repository in /home/tadarizhapov/laboratory/.git/
[tadarizhapov@tadarizhapov laboratory]$ █
```

рис.2

```
[tadarizhapov@tadarizhapov laboratory]$ ssh-keygen -C "Timur Darizhapov <timaanddar@mail.ru>"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tadarizhapov/.ssh/id_rsa): key
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in key.
Your public key has been saved in key.pub.
The key fingerprint is:
SHA256:waWKzD9UgkFbMs3o+54a8hzhDwfXy/A1P6WjleG2oMg Timur Darizhapov <timaanddar@mail.ru>
The key's randomart image is:
+----[RSA 2048]-----+
|  .+=.  .          |
|  .Bo. o          |
|  .o . =          |
|  o.. + o         |
|  *. = S . o . .  |
|  ..O . + o + =   |
|  . +. + . X      |
|  + o. + . . = +   |
|  +oo E . . .     |
+----[SHA256]-----+
```

рис.3

Регистрируемся на github.com. Переходим в Settings – SSH keys.

Вставляем в поле наш публичный ключ с помощью команды `cat ~/.ssh/id_rsa.pub`

[SSH keys](#) / Add new

Title

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', or 'ssh-ed25519'

рис.4



1

SHA256:N3/mj17NZQ430KIUAxcLs2W5UQ5V87Q4E0f16f4RvBo

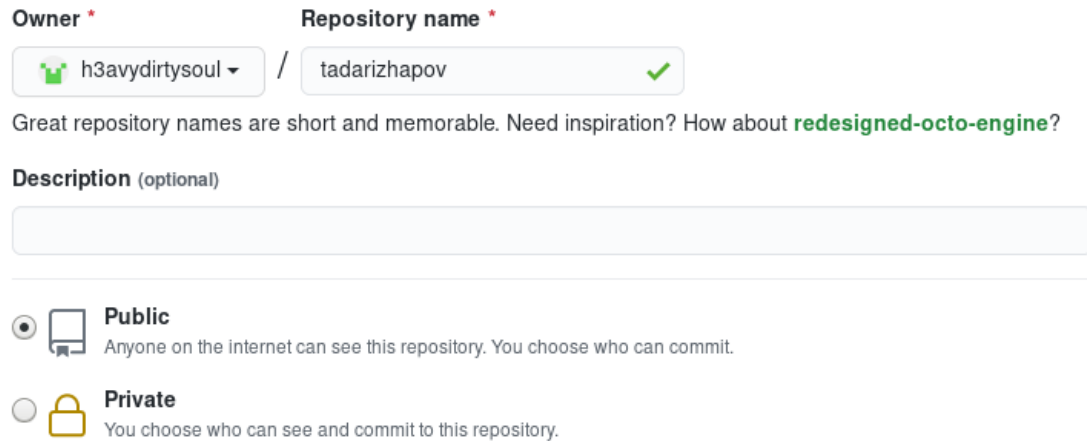
Added on 1 May 2021

Never used — Read/write

Delete

рис.5

2. Создаём репозиторий на сайте github.com. Делаем его публичным.



Owner * Repository name *

h3avydirty soul / tadarizhapov ✓

Great repository names are short and memorable. Need inspiration? How about **redesigned-octo-engine**?

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

рис.6

Создаём README.md. Печатаем в нём описание. Делаем первый commit.

```
[tadarizhapov@tadarizhapov laboratory]$ echo "#Лабораторные работы" >> README.md
[tadarizhapov@tadarizhapov laboratory]$ git add README.md
[tadarizhapov@tadarizhapov laboratory]$ git commit -m "my first commit"
[master (root-commit) cb69ee0] my first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
[tadarizhapov@tadarizhapov laboratory]$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       key
#       key.pub
nothing added to commit but untracked files present (use "git add" to track)
```

рис.7

Отправляем на github.com.

```
[tadarizhapov@tadarizhapov laboratory]$ git push -u origin master
Username for 'https://github.com': h3avydirty soul
Password for 'https://h3avydirty soul@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 252 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/h3avydirty soul/tadarizhapov/pull/new/master
remote:
To https://github.com/h3avydirty soul/tadarizhapov.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

рис.8

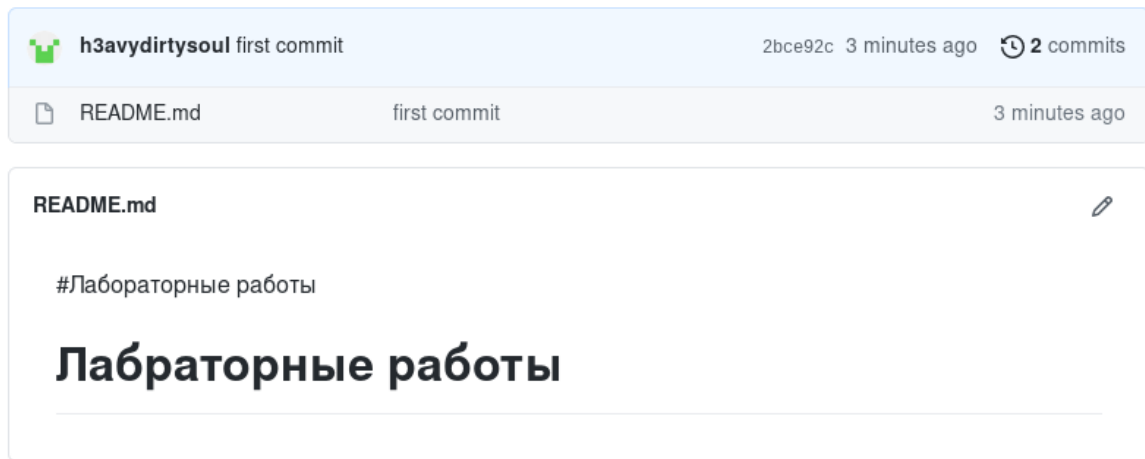


рис.9

3. Добавляем файл лицензии и шаблон игнорируемых файлов.

```
[tadarizhapov@tadarizhapov laboratory]$ wget https://creativecommons.org/licenses/by/4.0/legalcode.txt
--2021-05-01 10:21:23-- https://creativecommons.org/licenses/by/4.0/legalcode.txt
Распознаётся creativecommons.org (creativecommons.org)... 172.67.34.140, 104.20.151.16, 104.20.150.16, ...
Подключение к creativecommons.org (creativecommons.org)|172.67.34.140|:443... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа... 200 OK
Длина: нет данных [text/plain]
Сохранение в: «legalcode.txt»

[ <=> ] 18 657 --.-K/s за 0s

2021-05-01 10:21:24 (73,6 MB/s) - «legalcode.txt» сохранён [18657]
```

рис.10

Добавляем два новых файла, совершаем commit и отправляем на сайт.

```
[tadarizhapov@tadarizhapov laboratory]$ .gitignore
bash: .gitignore: команда не найдена...
[tadarizhapov@tadarizhapov laboratory]$ curl -L -s https://www.gitignore.io/api/c >> .gitignore
[tadarizhapov@tadarizhapov laboratory]$ git add .
[tadarizhapov@tadarizhapov laboratory]$ git status
# On branch master
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   .gitignore
#       new file:   key
#       new file:   key.pub
#       new file:   legalcode.txt
#
```

рис.11

```
[tadarizhapov@tadarizhapov laboratory]$ git push
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

    git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

    git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Username for 'https://github.com': h3avydirtysoul
Password for 'https://h3avydirtysoul@github.com':
Counting objects: 7, done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 8.19 KiB | 0 bytes/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/h3avydirtysoul/tadarizhapov.git
  2bce92c..e5b3ef5  master -> master
```

рис.12

4. В первой лабораторной работе было сказано установить образ операционной системы CentOS. В лабораторной работе есть команды только для Gentoo и Ubuntu. Ни одна из этих команд не помогла мне установить git flow. Посмотрев в интернете, решения этой проблемы я не нашёл.

```
[tadarizhapov@tadarizhapov laboratory]$ emerge dev-vcs/git-flow
bash: emerge: команда не найдена...
Аналогичная команда: 'merge'
[tadarizhapov@tadarizhapov laboratory]$ apt-get install git-flow
bash: apt-get: команда не найдена...
[tadarizhapov@tadarizhapov laboratory]$ sudo apt git-flow
[sudo] пароль для tadarizhapov:
sudo: apt: command not found
[tadarizhapov@tadarizhapov laboratory]$ apt-get install git-flow
bash: apt-get: команда не найдена...
```

рис.13

Вывод: я изучил применение средств контроля версий и их идеологию, научился на начальном уровне пользоваться git

3. Контрольные вопросы:

1) Система контроля версий (Version Control System, VCS) – 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. 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. 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.

2) Хранилище (репозиторий) – в нем хранятся все документы вместе с историей их изменения и другой служебной информацией. Рабочую копию необходимо периодически синхронизировать с репозиторием, эта операция предполагает отправку в него изменений, которые пользователь внес в свою рабочую копию. За это отвечает команда `commit`. С помощью этой команды можно также передать комментарий о сделанных изменениях. История – список предыдущих изменений/коммитов. Рабочая копия – копия проекта, связанная с репозиторием, в которой непосредственно работает пользователь.

3) Централизованные VCS Клиент-серверная модель: один центральный репозиторий, с которым разработчики взаимодействуют по сети.

Примеры:

- CVS
- Subversion (SVN)
- Perforce

Децентрализованная VCS

Клиенты полностью копируют репозиторий, вместо простого

скачивания снимка всех файлов (состояния файлов в определенный момент времени). В этом случае, если сервер выйдет из строя, то клиентский репозиторий можно будет скопировать на другой, рабочий, сервер, ведь данный репозиторий является полным бэкапом всех данных.

Примеры:

- Git
- Mercurial
- Bazaar

4) Действия с VCS при единоличной работе с хранилищем:

- Создать новый репозиторий на локальном устройстве (если он не был создан),

- Внести изменения в исходные файлы,
- Выполнить индексацию необходимых файлов,
- Проверить внесенные изменения,
- Выполнить commit,
- Отправить локальный репозиторий на удаленный сервер, при

необходимости.

5) Действия при работе с общим хранилищем VCS:

- Обычно проект уже создан и его нужно загрузить из общего удаленного хранилища,

- Необходимо создать свою рабочую ветку,
- Внести изменения внутри своей рабочей ветки,
- Выполнить индексацию необходимых файлов на своем локальном

устройстве,

- Проверить внесенные изменения,
- Выполнить commit,
- Свою рабочую ветку отправить в общее хранилище,
- При необходимости внести изменения и отправить снова,
- После администратор объединит вашу ветку с master branch.

6) 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. Git запоминает не все изменения, а только те, которые вы скажите. Обычно работа с Git выглядит так:

- сверстали шапку сайта – сделали commit;
- сверстали контент страницы – сделали второй commit;
- закончили верстать страницу – сделали третий commit и отправили код на сервер, чтобы вашу работу могли увидеть коллеги, либо чтобы опубликовать страницу с помощью Capistrano.

7) 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 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`

8) Примеры использования VCS при работе с локальными репозиториями: Создание небольшого проекта на своем локальном устройстве, работа с файлами (например, каталог, содержащий документы, презентации, которые будут часто редактироваться), работа с фотографиями, видео и т.д.

Примеры использования VCS при работе с удаленными репозиториями: Примеры могут быть те же, но теперь над ними работают несколько человек. Такая система позволяет следить за работой других пользователей.

9) Ветвление («ветка», `branch`) – один из параллельных участков истории в одном хранилище, исходящих из одной версии (точки ветвления).

- Обычно есть главная ветка (`master`), или ствол (`trunk`).
- Между ветками, то есть их концами, возможно слияние. Ветки используются для разработки одной части функционала изолированно от других. Каждая ветка представляет собой отдельную копию кода проекта. Ветки позволяют одновременно работать над разными версиями проекта.

Каждый репозиторий по-умолчанию имеет ветку master. Всякий раз, когда требуется разработка нового функционала, не внося при этом изменений в основную рабочую версию, можно создавать новую ветку, на основе рабочей, и вести разработку в ней – новой копии кода проекта. Когда функционал доделан и протестирован, можно сделать merge – слить отдельную ветку обратно с основной. При слиянии этой временной ветки в основную, все её коммиты разом перенесутся из одной в другую. Ветвление позволяет обеспечить процесс, при котором всегда в наличии будет рабочая версия проекта, в которой не будет частично завершённого функционала находящегося в активной разработке или же непротестированных фич.

10) 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`