

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

## Лабораторная работа № 3

дисциплина: *Операционные системы*

Студент: Мухамедияр Адиль

Группа: НКНбд-01-20

### Цель работы:

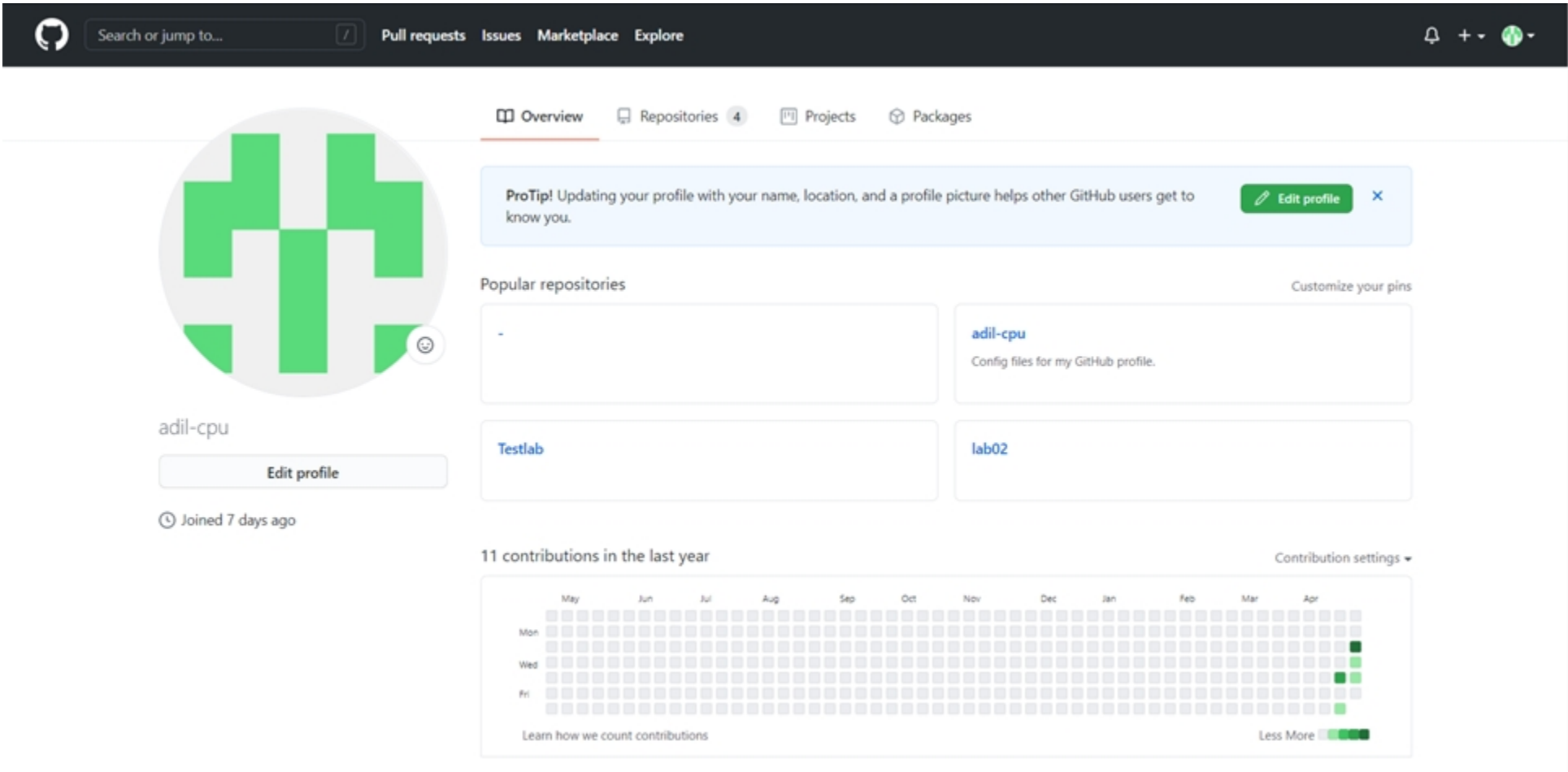
Научиться оформлять отчёты с помощью легковесного языка разметки Markdown.

### Выполнение задания

Оформление лабораторной работы №2 на Markdown.

### Ход работы:

1. Создал учетную запись на github.



2. Обозначил рабочий каталог как test, создав ее командой mkdir. После перешел в данный каталог командой cd. Инициализировал систему git командой git init. Создаю заготовку для файла README.md: echo "# lab02" >> README.md git add README.md – Делаю первый коммит и выкладываем на github: git commit -m "first commit" git remote add origin [git@github.com:sciproc-intro](https://github.com/sciproc-intro).git git push -u origin master

```

[amukhamediyar@amukhamediyar ~]$ mkdir test
[amukhamediyar@amukhamediyar ~]$ cd test
[amukhamediyar@amukhamediyar test]$ echo "# lab02" >> README.md
[amukhamediyar@amukhamediyar test]$ git init
Initialized empty Git repository in /home/amukhamediyar/test/.git/
[amukhamediyar@amukhamediyar test]$ git add README.md
[amukhamediyar@amukhamediyar test]$ git commit -m "first commit"
[master (root-commit) 18b2760] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
[amukhamediyar@amukhamediyar test]$ git branch -M main
[amukhamediyar@amukhamediyar test]$ git remote add origin git@github.com:adil-cpu/lab02.git
[amukhamediyar@amukhamediyar test]$ git push -u origin main
The authenticity of host 'github.com (140.82.121.4)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWGL7ElIGOCspRomTxdCARLviKw6E5SY8.
RSA key fingerprint is MD5:16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added 'github.com,140.82.121.4' (RSA) to the list of known hosts.
Counting objects: 3, done.
Writing objects: 100% (3/3), 217 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:adil-cpu/lab02.git
 * [new branch]      main -> main
Branch main set up to track remote branch main from origin.

```

3. Добавил файл лицензий:

```

[amukhamediyar@amukhamediyar test]$ wget https://creativecommons.org/licenses/by/4.0/legalcode.txt
--2021-04-29 20:13:39--  https://creativecommons.org/licenses/by/4.0/legalcode.txt
Распознаётся creativecommons.org (creativecommons.org)... 104.20.151.16, 104.20.150.16,
172.67.34.140, ...
Подключение к creativecommons.org (creativecommons.org)|104.20.151.16|:443... соединени
е установлено.
HTTP-запрос отправлен. Ожидание ответа... 200 OK
Длина: нет данных [text/plain]
Сохранение в: «legalcode.txt»

[ <=> ] 18 657 --.-K/s за 0,001s

2021-04-29 20:13:41 (14,6 MB/s) - «legalcode.txt» сохранён [18657]

```

4. Для начало посмотрим список имеющихся шаблонов

```
[amukhamediyar@amukhamediyar test]$ curl -L -s https://www.gitignore.io/api/list
lc,lc-bitrix,a-frame,actionsript,ada
adobe,advancedinstaller,adventuregamestudio,agda,al
alteraquartusii,altium,amplify,android,androidstudio
angular,anjuta,ansible,apachecordova,apachehadoop
appbuilder,appcelerator titanium,appcode,appcode+all,appcode+iml
appengine,aptanastudio,arcanist,archive,archives
archlinuxpackages,aspnetcore,assembler,ate,atmelstudio
ats,audio,automationstudio,autotools,autotools+strict
awr,azurefunctions,backup,ballerina,basercms
basic,batch,bazaar,bazel,bitrise
bitrix,bittorrent,blackbox,bloop,bluej
bookdown,bower,bricxcc,buck,c
c++,cake,cakephp,cakephp2,cakephp3
calabash,carthage,certificates,ceylon,cfwheels
chefcookbook,chocolatey,clean,clion,clion+all
clion+iml,clojure,cloud9,cmake,cocoapods
cocos2dx,cocoscreator,code,code-java,codeblocks
codecomposerstudio,codeigniter,codeio,codekit,codesniffer
coffeescript,commonlisp,compodoc,composer,compressed
compressedarchive,compression,conan,concrete5,coq
cordova,craftcms,crashlytics,crbasic,crossbar
crystal,cs-cart,csharp,cuda,cvs
cypressio,d,dart,darteditor,data
database,datarecovery,dbeaver,defold,delphi
dframe,diff,direnv,diskimage,django
```

---

5. Теперь скачиваем шаблон для C

```
[amukhamediyar@amukhamediyar test]$ curl -L -s https://www.gitignore.io/api/c >> .gitignore
[amukhamediyar@amukhamediyar test]$ ls
legalcode.txt  README.md
```

6. Добавил новые файлы и выполнил коммит

```
[amukhamediyar@amukhamediyar test]$ git add .
[amukhamediyar@amukhamediyar test]$ git commit -m "push"
[main 9a8a58d] push
2 files changed, 455 insertions(+)
create mode 100644 .gitignore
create mode 100644 legalcode.txt
```

7. Отправим на github



Файл Правка Вид Поиск Терминал Справка

```
[amukhamediyar@amukhamediyar test]$ 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)

Warning: Permanently added the RSA host key for IP address '140.82.121.3' to the list o
f known hosts.
Counting objects: 5, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 6.43 KiB | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To git@github.com:adil-cpu/lab02.git
    18b2760..9a8a58d  main -> main
```

8. Инициализировал git-flow, установив префикс для ярлыков в v.

```
[amukhamediyar@amukhamediyar test]$ git flow init

Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? [] v
```

9. Проверяю, что я нахожусь на ветке develop.

```
[amukhamediyar@amukhamediyar test]$ git branch
* develop
  main
```

10. Создаю релиз с версией 1.0.0

```
[amukhamediyar@amukhamediyar test]$ git flow release start 1.0.0
Switched to a new branch 'release/1.0.0'
```

Summary of actions:

- A new branch 'release/1.0.0' was created, based on 'develop'
- You are now on branch 'release/1.0.0'

Follow-up actions:

- Bump the version number now!
- Start committing last-minute fixes in preparing your release
- When done, run:

```
git flow release finish '1.0.0'
```

11. Записал версию

```
[amukhamediyar@amukhamediyar test]$ echo "1.0.0" >> VERSION
[amukhamediyar@amukhamediyar test]$ ls
legalcode.txt  README.md  VERSION
```

12. Добавил в индекс

```
[amukhamediyar@amukhamediyar test]$ git add *
[amukhamediyar@amukhamediyar test]$ git commit -m "first version"
[release/1.0.0 b96c270] first version
1 file changed, 1 insertion(+)
create mode 100644 VERSION
```

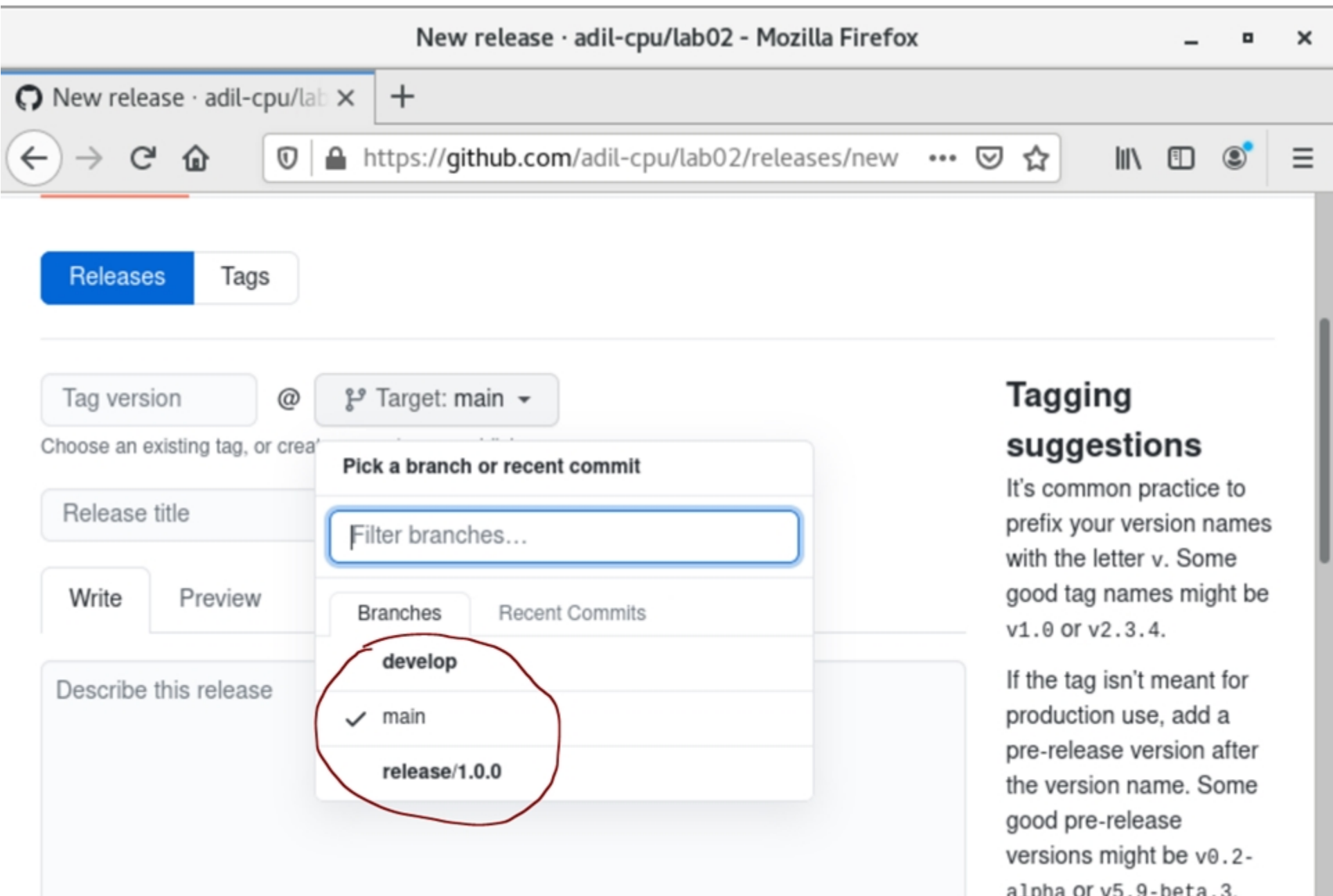
13. Залил релизную ветку в основную ветку

```
[amukhamediyar@amukhamediyar test]$ git flow release finish 1.0.0
Switched to branch 'main'
Merge made by the 'recursive' strategy.
VERSION | 1 +
1 file changed, 1 insertion(+)
create mode 100644 VERSION
fatal: no tag message?
Tagging failed. Please run finish again to retry.
```

14. Отправил данные на github

```
[amukhamediyar@amukhamediyar test]$ git push --all
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 368 bytes | 0 bytes/s, done.
Total 4 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To git@github.com:adil-cpu/lab02.git
    9a8a58d..b545d61  main -> main
* [new branch]      develop -> develop
* [new branch]      release/1.0.0 -> release/1.0.0
[amukhamediyar@amukhamediyar test]$ git push --tags
Everything up-to-date
```

15. Проверил



## Вывод:

Изучил и понял как работать с системой контроля версий, с помощью командной строки, а именно с Git. Разобрал команды.

Контрольные вопросы:

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначены?

Система контроля версий (Version Control System, VCS) представляет собой программное обеспечение для облегчения работы с изменяющейся информацией. VCS нужны для хранения полной истории изменений; Описания причин всех производимых изменений; Отката изменений, если что-то пошло не так; Поиска причины и ответственного за появления ошибок в программе; Совместной работы группы над одним проектом; Возможности изменять код, не мешая работе других пользователей.

2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище (repository), или репозиторий, — место хранения файлов и их версий, служебной информации. Версия (revision), или ревизия, — состояние всего хранилища или отдельных файлов в момент времени («пункт истории»). Commit («[трудовой] вклад», не переводится) — процесс создания новой версии; иногда синоним версии. Рабочая копия (working copy) — текущее состояние файлов проекта (любой версии), полученных из хранилища и, возможно, измененных.

3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. Централизованные системы контроля версий представляют собой приложения типа клиент-сервер, когда репозиторий проекта существует в единственном экземпляре и хранится на сервере. Доступ к нему осуществлялся через специальное клиентское приложение. Пример: CVS-одна из первых систем второго поколения (1986г.). Обладает множеством недостатков и считается устаревшей.



Децентрализованные системы контроля версий, в отличие от централизованной модели, может существовать несколько экземпляров репозитория, которые время от времени синхронизируются между собой. Пример: Git- распределенная система управления версиями, созданная Л. Торвальдсом для управления разработкой ядра Linux.

Отличия между централизованными и децентрализованными VCS. Централизованные:

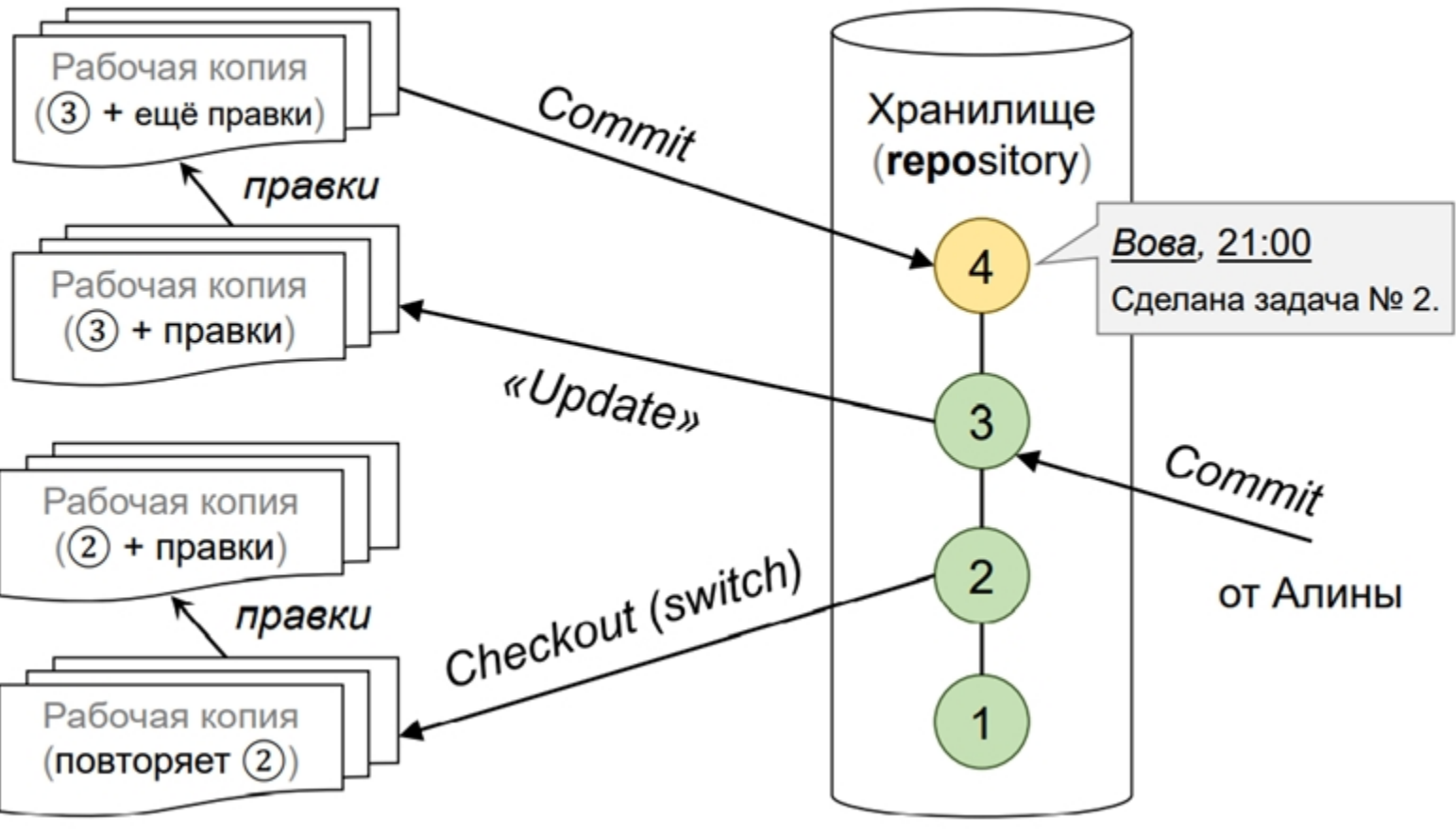
- Простота использования.
- Вся история — всегда в едином общем хранилище.
- Нужно подключение к сети.
- Резервное копирование нужно только одному хранилищу.
- Удобство разделения прав доступа к хранилищу.
- Почти все изменения навсегда попадают в общее хранилище. Децентрализованные:
- Двухфазный commit:
  - 1. запись в локальную историю;
  - 2. пересылка изменений другим.
- Подключение к сети не нужно.
- Локальные хранилища могут служить резервными копиями.
- Локальное хранилище контролирует его владелец,
- но общее — администратор.

• Возможна правка локальной истории перед отправкой на сервер.

4. Опишите действия с VCS при единоличной работе с хранилищем. 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.

5. Опишите порядок работы с общим хранилищем VCS. Работа с общим хранилищем выглядит так:

# Работа с общим хранилищем



6. Каковы основные задачи, решаемые инструментальным средством git? Задачи решаемые git: Как не потерять файлы с исходным кодом? Как защититься от случайных исправлений и удалений? Как отменить изменения, если они оказались некорректными? Как одновременно



поддерживать рабочую версию и разработку новой?

7. Назовите и дайте краткую характеристику командам git.

- add - добавить файл или папку в репозиторий git;
- am - применить все патчи из email;
- archive - создать архив файлов;
- bisect - использовать бинарный поиск для поиска нужного коммита;
- branch - управление ветками проекта;
- bundle - перемещение объектов и ссылок в архиве;
- checkout - переключение между ветками;
- cherry-pick - внести изменения в уже существующие коммиты;
- clean - удалить все неотслеживаемые файлы и папки проекта;
- clone - создать копию удаленного репозитория в папку;
- commit - сохранить изменения в репозиторий;
- diff - посмотреть изменения между коммитами;
- fetch - скачать удаленный репозиторий;
- init - создать репозиторий;
- merge - объединить две ветви;
- pull - интегрировать удаленный репозиторий с локальным;
- push - отправить изменения в удаленный репозиторий;
- tag - управление тегами;
- worktree - управление деревьями разработки.

8. Приведите примеры использования при работе с локальным и удалённым репозиториями.

9. Что такое и зачем могут быть нужны ветви (branches)? Ветки нужны для того, чтобы программисты могли вести совместную работу над проектом и не мешать друг другу при этом. При создании проекта, Git создает базовую ветку. Она называется master веткой. Она считается центральной веткой, т.е. в ней содержится основной код приложения.

10. Как и зачем можно игнорировать некоторые файлы при commit? Игнорируемые файлы — это, как правило, специфичные для платформы файлы или автоматически созданные файлы из систем сборки. Некоторые общие примеры включают в себя: Файлы времени выполнения, такие как журнал, блокировка, кэш или временные файлы. Файлы с конфиденциальной информацией, такой как пароли или ключи API. Скомпилированный код, такой как .class или .o. Каталоги зависимостей, такие как /vendor или /node\_modules. Создавать папки, такие как /public, /out или /dist. Системные файлы, такие как .DS\_Store или Thumbs.db Конфигурационные файлы IDE или текстового редактора.

---

## Вывод к 3 лабораторной работе.

Я изучив базовые сведения о Markdown, научился оформлять отчеты. Освоил синтаксис данного языка разметки. Выполнил 2 лабораторную работу на Markdown.

---