ИНСТРУКЦИЯ ИСПОЛЬЗОВАНИЯ БИБЛИОТЕКИ

ШАГ 0 – Открыть терминал

В5кВ Ubuntu консоль запускается при загрузке системы. Терминал – это тоже консоль, но уже в графической оболочке. Его можно запустить, набрав слово Терминал в поисковой строке ОС, или через комбинацию клавиш **Ctrl+Alt+T**.

В общем виде в Ubuntu команды имеют такой вид:

<программа – ключ значение>

Программа – это сам исполняемый файл. Другими словами, это программа, которая будет выполняться по команде.
Ключ – обычно у каждой программы свой набор ключей. Их можно найти в мануале к программе.
Значение – параметры программы: цифры, буквы, символы, переменные.

Напомним, что для выполнения команды нужно ввести её в командную строку – Ubuntu console или эмулирующий работу консоли терминал.

Рассмотрим основные команды консоли Ubuntu:

<sudo>
Промежуточная команда **sudo (SuperUser DO – суперпользователь)** позволяет запускать программы от имени администратора или root-пользователя. Вы можете добавить sudo перед любой командой, чтобы запустить её от имени суперпользователя.

<apt>
Команда **apt** используется для работы с программными пакетами для установки программных пакетов (sudo apt install имя-пакета), обновления репозитория с пакетами (sudo apt update) и обновления пакетов, которые установлены в систему (sudo apt upgrade).

<pwd>
Команда **pwd (print working directory – вывести рабочую директорию)** показывает полное имя рабочей директории, в которой вы находитесь.

<ls>
Команда **ls (list – список)** выводит все файлы во всех папках рабочей директории. С помощью ls -a можно вывести и скрытые файлы.

<cd>
Команда **cd (change directory – изменить директорию)** позволяет перейти в другую директорию. Можно ввести как полный путь до папки, так и её название. Например, чтобы попасть в папку Files, лежащую в директории /user/home/Files, введите cd Files или cd /user/home/Files. Чтобы попасть в корневую директорию, введите cd /.

<cp>
Команда **cp (copy – копировать)** копирует файл. Например, cp file1 file2 скопирует содержимого файла file1 в file2. Команда cp file /home/files скопирует файл с названием file в директорию /home/files.

<mv>
Команда **mv (move – переместить)** помогает перемещать файлы. Также с помощью mv можно переименовывать файлы. Например, у нас есть файл file.txt. С помощью команды mv

file.txt new_file.txt мы можем перенести его в ту же директорию, но у файла уже будет новое название new_file.txt.

<rm>
Команда **rm (remove – удалить)** удаляет файлы и каталоги. Так, команда rm file.txt удалит текстовый файл с названием file, а команда rm –r Files удалит директорию Files со всеми содержащимися в ней файлами.

<mkdir>
С помощью **mkdir (make directory – создать директорию)** можно создать новую директорию. Так, команда mkdir directory создаст новую директорию с именем directory в текущей рабочей директории.

<man>
Команда **man (manual – мануал)** открывает справочные страницы с подробной информацией о команде. Введите man, а затем через пробел название команды, о которой вы хотите узнать подробнее. Например, man cp выведет справочную страницу о команде cp.

ШАГ 1 – Установить зависимости

```
sudo apt install build-essential
sudo apt install libgoogle-glog-dev
sudo apt install cmake
sudo apt install git
```

ШАГ 2 – Создать и перейти в удобный каталог

```
mkdir testing
cd testing
```

ШАГ 3 – Склонировать репозитории OpenCV и перейти на коммит версии 4.6.0

```
git clone https://github.com/opencv/opencv.git
cd opencv
git checkout 4.6.0
cd ..
git clone https://github.com/opencv/opencv_contrib.git
cd opencv_contrib
git checkout 4.6.0
cd ..
```

ШАГ 4 – Произвести сборку и установку OpenCV

```
cd opencv
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=/usr/local
-DWITH_TBB=ON -DWITH_V4L=ON -DWITH_QT=ON -DWITH_OPENGL=ON -
DOPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules ..
make
sudo make install
cd ../..
```

ШАГ 5 – Склонировать репозиторий библиотеки tclap

```
git clone https://github.com/mirror/tclap.git
```

ШАГ 6 – Склонировать репозиторий проекта

git clone https://github.com/ChervyakovLM/FaceMetric.git

ШАГ 7 – Произвести сборку

```
cd FaceMetric
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=../Release -
DTCLAP_INCLUDE_DIR=/home/{USER}/testing/tclap/include -
DFACE_API_ROOT_DIR=/home/{USER}/testing/FaceMetric/CI/face_api_test -
DFREEIMAGE_ROOT_DIR=/home/{USER}/testing/FaceMetric/CI/FreeImage ..
make
make install
cd ..
```

ШАГ 8 – В каталоге проекта FaceMetric появится папка Release, в которой будут находить исполняемые файлы для проверки верификации и идентификации.

ШАГ 9 – Для запуска верификации необходимо выполнить команду

./checkFaceApi_V –split=./verification

checkFaceApi_V **имеет следующие флаги**:
– **split** – путь до каталога с тестовыми данными, необходимый параметр
– **config** – путь до каталога с конфигурационными файлами FaceEngine, по-умолчанию: input/config
– **extract_list** – путь до списка извлечённых файлов, по-умолчанию: input/extract.txt
– **extract_prefix** – путь до каталога с изображениями, по-умолчанию: input/images
– **grayscale** – открывать изображения как оттенки серого, по-умолчанию: false
– **count_proc** – число используемых ядер процессора, по-умолчанию: thread::hardware_concurrency()
– **extra_timings** - расширенная временная статистика, по-умолчанию: false
– **extract_info** - логирование дополнительных параметров экстракции признаков, по-умолчанию: false
– **debug_info** – вывод отладочной информации, по-умолчанию: false
– **desc_size** - размер дескриптора, по-умолчанию: 512
– **percentile** - параметр управления временной статистики в %, по-умолчанию: 90
– **do_extract** – стадия извлечения признаков из изображений, по-умолчанию: true
– **do_match** – стадия сравнения признаков друг с другом, по-умолчанию: true
– **do_ROC** – стадия расчёта точек ROC-кривой, по-умолчанию: true

ШАГ 10 – Для запуска идентификации необходимо выполнить команду

./checkFaceApi_I –split=./identification

checkFaceApi_I **имеет следующие флаги**:
– **split** - путь до каталога с тестовыми данными, необходимый параметр
– **config** - путь до каталога с конфигурационными файлами FaceEngine, по-умолчанию: input/config
– **db_list** – путь до базы данных, списка индексов, по-умолчанию: input/db.txt
– **mate_list** - список запросов для лиц, которые есть в базе, по-умолчанию: input/mate.txt
– **nonmate_list** - список запросов для лиц которых нет в базе, по-умолчанию: input/nonmate.txt
– **insert_list** - список для всавки в базу, по-умолчанию: input/insert.txt

– **remove_list** - список для удалении из базы, по-умолчанию: input/remove.txt
– **extract_prefix** - путь до каталога с изображениями, по-умолчанию: input/images
– **grayscale** - открывать изображения как оттенки серого, по-умолчанию: false
– **count_proc** – число используемых ядер процессора, по-умолчанию: thread::hardware_concurrency()
– **extra_timings** - расширенная временная статистика, по-умолчанию: false
– **extract_info** - логирование дополнительных параметров экстракции признаков, по-умолчанию: false
– **debug_info** – вывод отладочной информации, по-умолчанию: false
– **desc_size** - размер дескриптора, по-умолчанию: 512
– **percentile** - параметр управления временной статистики в %, по-умолчанию: 90
– **nearest_count** - максимальное количество кандидатов для поиска в базе, false, 100
– **search_info** - логирование дополнительных результатов поиска, по-умолчанию: false
– **do_extract** - стадия извлечения признаков из изображений, по-умолчанию: true
– **do_graph** – стадия преобразования признаков изображения в индекс, по-умолчанию: true
– **do_insert** – стадия добавления в индекс, по-умолчанию: true
– **do_remove** – стадия удаления из индекса, по-умолчанию: true
– **do_search** – стадия поиска в индексе, по-умолчанию: true
– **do_tpir** – стадия расчёта метрик индетификации, по-умолчанию: true

Пример №1 FACEAPITEST: Interface
Для проверки заданной библиотеки биометрической верификации необходимо реализовать класс наследник от FACEAPITEST: Interface.

```
class Interface {
public:
virtual ~Interface() {}
virtual ReturnStatus
initialize(const std::string &configDir) = 0;
virtual ReturnStatus
createTemplate(
const Multiface &faces,
TemplateRole role,
std::vector<uint8_t> &templ,std::vector<EyePair> &eyeCoordinates,
std::vector<double> &quality) = 0;
virtual ReturnStatus
matchTemplates(
const std::vector<uint8_t> &verifTemplate,
const std::vector<uint8_t> &initTemplate,
double &similarity) = 0;
virtual ReturnStatus
train(
const std::string &configDir,
const std::string &trainedConfigDir) = 0;
static std::shared_ptr<Interface>
getImplementation();
};
```

Класс наследник должен содержать реализацию следующих функций:
initialize - инициализация алгоритма вычисления биометрических шаблонов;
createTemplate - вычисление шаблона;
matchTemplates - сравнение шаблонов;
train - донастройка алгоритма вычисления биометрических шаблонов;

getImplementation – получение указателя на реализацию.

Пример реализации класса наследника приведён в файлах face_api_example_V.h и face_api_example_V.cpp, содержащиеся в каталогах include и src соответственно.

Пример №2 FACEAPITEST: IdentInterface
Для проверки заданной библиотеки биометрической идентификации необходимо реализовать класс наследник от FACEAPITEST: IdentInterface.

```
class IdentInterface {
public:
virtual ~IdentInterface() {}
virtual ReturnStatus
initializeTemplateCreation(
const std::string &configDir,
TemplateRole role) = 0;
virtual ReturnStatus
createTemplate(
const Multiface &faces,
TemplateRole role,
std::vector<uint8_t> &templ,
std::vector<EyePair> &eyeCoordinates) = 0;
virtual ReturnStatus
finalizeInit(
const std::string &configDir,
const std::string &initDir,
const std::string &edbName,
const std::string &edbManifestName) = 0;virtual ReturnStatus
initializeIdentification(
const std::string &configDir,
const std::string &initDir) = 0;
virtual ReturnStatus
identifyTemplate(
const std::vector<uint8_t> &idTemplate,
const uint32_t candidateListLength,
std::vector<Candidate> &candidateList,
bool &decision) = 0;
virtual ReturnStatus
galleryInsertID(
const std::vector<uint8_t> &templ,
const std::string &id) = 0;
virtual ReturnStatus
galleryDeleteID(
const std::string &id) = 0;
static std::shared_ptr<IdentInterface>
getImplementation();
};
```

Класс наследник должен содержать реализацию следующих функций:

initializeTemplateCreation – инициализация алгоритма вычисления биометрических шаблонов;
createTemplate - вычисление шаблона;
finalizeInit - создание индекса из всех шаблонов;

initializeIdentification - инициализация алгоритма поиска по индексу;
identifyTemplate – поиск по индексу;
galleryInsertID - добавление шаблона в индекс;
galleryDeleteID - удаление шаблона из индекса;
getImplementation – получение указателя на реализацию.

Пример реализации класса наследника приведён в файлах face_api_example_I.h и face_api_example_I.cpp, содержащиеся в каталогах include и src соответственно.