LIBRARY OPERATING INSTRUCTIONS

Step 0 – Open terminal

In Ubuntu, the console starts when the system boots. The terminal is also a console, but already in a graphical shell. It can be launched by typing the word Terminal in the OS search bar, or through the key combination **Ctrl + Alt + T**.

In general, in Ubuntu, the commands are as follows:

<program – key value>

The program is the executable itself. In other words, a program will be executed on command.
Key – usually each program has its own set of keys. They can be found in the manual for the program.
Value – program parameters: digits, letters, symbols, variables.

Recall that to execute a command, you need to enter it into the command line – Ubuntu console or a terminal that emulates the operation of the console.

Consider the basic Ubuntu console commands:

<sudo>
The intermediate command **sudo** (SuperUser DO – superuser) allows you to run programs as an administrator or root user. You can add **sudo** before any command to run it as root.

<apt>
Command **apt** is used to work with software packages to install software packages (sudo apt install package–name), update a package repository (sudo apt update), and upgrade packages that are installed on the system (<sudo apt upgrade>).

<pwd>
Command **pwd** (print working directory) shows the full name of the working directory you are in.

<ls>
Command **ls** (list) displays all files in all folders of the working directory. You can also list hidden files with ls –a.

<cd>
Command **cd** (change directory) allows you to change directory. You can enter both the full path to the folder and its name. For example, to get to the Files folder in the /user/home/Files directory, type cd Files or cd /user/home/Files. To get into the root directory, type cd /.

<cp>
Command **cp** (copy) copies the file.
For example, cp file1 file2 will copy the contents of file1 to file2.
The cp file /home/files command will copy a file named file to the /home/files directory.

<mv>
Command **mv** (move) helps to move files. You can also rename files with mv. For example, we have a file file.txt. With the command mv file.txt new_file.txt we can move it to the same directory, but the file will already have a new name new_file.txt.

<rm>

Command **rm** (remove) deletes files and directories. For example, the rm file.txt command will delete the text file named file, and the rm -r Files command will delete the Files directory with all the files it contains.

<mkdir>
With **mkdir** (make directory) you can create a new directory. Thus, the mkdir directory command will create a new directory named directory in the current working directory.

<man>
Command **man** (manual) opens man pages with detailed information about the command. Type man followed by a space followed by the name of the command you want to learn more about. For example, man cp will display a man page for the cp command.

Step 1 – Install Dependencies

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

Step 2 – Create and change to a convenient directory

```
mkdir testing
cd testing
```

Step 3 – Clone OpenCV repositories and commit version 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 ..
```

Step 4 – Build and install 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 ../..
```

Step 5 – Clone tclap library repository

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

Step 6 – Clone project repository

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

Step 7 – Make an assembly

```
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 ..
```

Step 8 – The Release folder will appear in the FaceMetric project directory, in which executable files will be found for checking verification and identification.

Step 9 – To start verification, you need to run the command

```
./checkFaceApi_V –split=./verification
```

checkFaceApi_V **has the following flags**:
– **split** – path to the directory with test data, required parameter
– **config** – path to the directory with FaceEngine configuration files, by default: input/config
– **extract_list** – path to the list of extracted files, by default: input/extract.txt
– **extract_prefix** – path to the directory with images, by default: input/images
– **grayscale** – open images as grayscale, default: false
– **count_proc** – number of used processor cores, by default: thread: hardware_concurrency()
– **extra_timings** – extended timing statistics, default: false
– **extract_info** – logging additional parameters of feature extraction, default: false
– **debug_info** – display debug information, default: false
– **desc_size** – descriptor size, default: 512
– **percentile** – time statistics control parameter in %, default: 90
– **do_extract** – stage of feature extraction from images, default: true
– **do_match** – stage of feature comparison with each other, default: true
– **do_ROC** – stage of calculation of ROC–curve points, by default: true

Step 10 – To start identification, you need to execute the command

```
./checkFaceApi_I –split=./identification
```

checkFaceApi_I **has the following flags**:
– **split** – path to the directory with test data, required parameter
– **config** – path to the directory with FaceEngine configuration files, by default: input/config
– **db_list** – path to the database, list of indexes, by default: input/db.txt
– **mate_list** – a list of requests for persons that are in the database, by default: input/mate.txt
– **nonmate_list** – list of requests for persons who are not in the database, by default: input/nonmate.txt
– **insert_list** – list to be inserted into the database, by default: input/insert.txt
– **remove_list** – list to be removed from the database, by default: input/remove.txt
– **extract_prefix** – path to the directory with images, by default: input/images
– **grayscale** – open images as grayscale, default: false
– **count_proc** – number of processor cores used, by default: thread::hardware_concurrency()
– **extra_timings** – extended timing statistics, default: false
– **extract_info** – logging additional parameters of feature extraction, default: false
– **debug_info** – display debug information, default: false
– **desc_size** – descriptor size, default: 512
– **percentile** – time statistics control parameter in %, default: 90

– **nearest_count** – maximum number of candidates to search in the database, false, 100
– **search_info** – logging additional search results, default: false
– **do_extract** – stage of feature extraction from images, default: true
– **do_graph** – stage of converting image features into an index, by default: true
– **do_insert** – stage of adding to the index, by default: true
– **do_remove** – stage of removal from the index, by default: true
– **do_search** – index search stage, default: true
– **do_tpir** – identification metrics calculation stage, default: true

Example #1 FACEAPITEST: Interface.
To check the given biometric verification library, it is necessary to implement a class that inherits from 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();
};
```

The inheritor class must contain the implementation of the following functions:
initialize – initialization of the algorithm for calculating biometric templates;
createTemplate – template calculation;
matchTemplates – template comparison;
train – additional adjustment of the algorithm for calculating biometric templates;
getImplementation – get a pointer to the implementation.

An example of the implementation of the successor class is given in the face_api_example_V.h and face_api_example_V.cpp files contained in the include and src directories, respectively.

Example #2 FACEAPITEST: IdentInterface.
To check the given biometric identification library, it is necessary to implement the class inherited from 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();
};
```

The inheritor class must contain the implementation of the following functions:
initializeTemplateCreation – initialization of the algorithm for calculating biometric templates;
createTemplate – template calculation;
finalizeInit – create an index from all templates;
initializeIdentification – initialization of the index search algorithm;
identifyTemplate – search by index;
galleryInsertID – adding a template to the index;
galleryDeleteID – removal of the template from the index;
getImplementation – get a pointer to the implementation.

An example of the implementation of the successor class is given in the face_api_example_I.h and face_api_example_I.cpp files contained in the include and src directories, respectively.