Modern C++ for Computer Vision and Image Processing

Igor Bogoslavskyi and Cyrill Stachniss
Outline

Course introduction

Linux introduction

C++ syntax

Hello World!
What you will learn in course

- How to work in Linux
- How to write software with modern C++
- Core software development techniques
- How to work with images using OpenCV
- How to implement inverse image search

Check out Google Image Search for example: https://images.google.com/
Why C++? Why Linux? Why?

- Over 50,000 developers surveyed
- Nearly half of them use Linux
- C++ is the most used systems language (4.5 million users in 2015)
- C++11 is a modern language
- All companies want C++ in our field

CLion survey: https://blog.jetbrains.com/clion/2015/07/infographics-cpp-facts-before-clion/
Course structure

The course is split in **two parts**:

1. **Learning the basics**
   - Consists of lectures and homeworks
   - 5 homeworks, 10 points each
   - 25 points moves you to the next part

2. **Working on a project**
   - Plan and code **inverse image search**
   - Groups of 2 — 3 people
   - Final project presentation in the end of semester
   - **Exam = project presentation**
Batteries included!

We will provide you with all the essential tools for the course:

- An **Ubuntu virtual machine**
- Lecture recordings on ECampus, YouTube
- Git server to store your code and submit homework assignments
What is Linux?

- Linux is a free **Unix-like OS**
- Linux kernel implemented by Linus Torvalds
- **Extremely popular:** Android, ChromeOS, servers, supercomputers, etc.
- Many **Linux distributions** available
- Use any distribution if you have preference
- Examples will be given in **Ubuntu**
Tree organization starting with root: /
- There are no volume letters, e.g. C:, D:
- User can only access his/her own folder
Understanding files and folders

- Folders end with / e.g. /path/folder/
- Everything else is files, e.g. /path/file
- Absolute paths start with / while all other paths are relative:
  - /home/igor/folder/ — absolute path to a folder
  - /home/igor/file.cpp — absolute path to a file
  - folder/file — relative path to a file

- Paths are case sensitive:
  filename is different from FileName
- Extension is part of a name:
  filename.cpp is different from filename.png
Linux terminal

- Press \textbf{Ctrl} + \textbf{Alt} + \textbf{T} to open terminal

- Linux terminal is a very powerful tool
- Most tasks can be done faster from the terminal than from the GUI
Navigating tree from terminal

- Terminal is always in some folder
- `pwd`: print working directory
- `cd <dir>`: change directory to `<dir>`
- `ls <dir>`: list contents of a directory
- Special folders:
  - `/` — root folder
  - `~` — home folder
  - `.` — current folder
  - `..` — parent folder
Structure of Linux commands

Typical structure

```
${PATH}/command [ options ] [ parameters ]
```

- `${PATH}/command`: absolute or relative path to the program binary
- `[options]`: program-specific options e.g. `-h`, or `--help`
- `[parameters]`: program-specific parameters e.g. input files, etc.
Use help with Linux programs

- **man** `<command>` — manual
  exhaustive manual on program usage
- **command** `-h`
  **command** `--help`
  usually shorter help message

```bash
igor@igor-lab:~$ pdfpc -h
pdfpc v3.1.1
Usage:
  pdfpc [OPTION...] <pdf-file>
Help Options:
  -h, --help           Show help options
Application Options:
  -d, --duration=N    Duration in minutes
```
Using command completion

Pressing \( \text{tab} \) while typing:
- completes name of a file, folder or program
- “beeps” if current text does not match any file or folder uniquely

Pressing \( \text{tab} + \) \( \text{tab} \) shows all potential matches

**Example:**

1. `igor@igor-work:~> cd te [TAB] [TAB]
2. teaching/ temp/ testing/`
Creating and manipulating files and folders

- `mkdir [-p] <foldername>` — make directory
  Create a folder `<foldername>` (with all parent folders [-p])

- `rm [-r] <name>` — remove [recursive]
  Remove file or folder `<name>`
  (With folder contents [-r])

- `cp [-r] <source> <dest>` — copy
  Copy file or folder from `<source>` to `<dest>`

- `mv <source> <dest>` — move
  Move file or folder from `<source>` to `<dest>`
## Using placeholders

<table>
<thead>
<tr>
<th>Placeholder</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Any set of characters</td>
</tr>
<tr>
<td>?</td>
<td>Any single character</td>
</tr>
<tr>
<td>[a–f]</td>
<td>Characters in [abcdef]</td>
</tr>
<tr>
<td>[^a–c]</td>
<td>Any character not in [abc]</td>
</tr>
</tbody>
</table>

Can be used with most of terminal commands: `ls`, `rm`, `mv` etc.
Example: placeholders

1 igor@igor-laptop:~/teaching/demo> ls
2 u01.tex u02.tex u03.tex v01_a.tex v01.pdf v01.tex
   v02.pdf v02.tex v03.pdf v03.tex
3 igor@igor-laptop:~/teaching/demo> ls *.pdf
4 v01.pdf v02.pdf v03.pdf
5 igor@igor-laptop:~/teaching/demo> ls u*
6 u01.tex u02.tex u03.tex
7 igor@igor-laptop:~/teaching/demo> ls ?01*
8 u01.tex v01_a.tex v01.pdf v01.tex
9 igor@igor-laptop:~/teaching/demo> ls [uv]01*
10 u01.tex v01_a.tex v01.pdf v01.tex
11 igor@igor-laptop:~/teaching/demo> ls u0[^12].tex
12 u03.tex
Standard input/output channels

- Single input channel: stdin
- Two output channels:
  - stdout: Standard output: channel 1
  - stderr: Standard error output: channel 2
- Redirecting stdout
  - command 1> out.txt
  - command >> out.txt
- Redirecting stderr
  - command 2> out.txt
- Redirect stdout and stderr into a file
  - programm > out.txt 2>&1
- Write stdout and stderr into different files
  - programm 1>stdout.txt 2>stderr.txt
Working with files

- **more/less/cat** `<filename>`
  Print the contents of the file
  Most of the time using *cat* if enough

- **find** `<in-folder>` `-name` `<filename>`
  Search for file `<filename>` in folder `<in-folder>`, allows wildcards

- **grep** `<what>` `<where>`
  Search for a string `<what>` in a file `<where>`
Chaining commands

- `command1; command2; command3`
  Calls commands one after another

- `command1 && command2 && command3`
  Same as above but fails if any of the commands returns a non-zero code

- `command1 | command2 | command3`
  **Pipe** stdout of `command1` to stdin of `command2` and stdout of `command2` to stdin of `command3`

  Piping commonly used with **grep**:
  `ls | grep smth` look for `smth` in output of `ls`
Canceling commands

- **CTRL + C**
  Cancel currently running command

- **kill -9 <pid>**
  Kill the process with id **pid**

- **killall <pname>**
  Kill all processes with name **pname**

- **htop (top)**
  - Shows an overview of running processes
  - Allows to kill processes by pressing **F9**
Command history

The shell saves the history of commands in the 
`~/.bash_history` file

- `↑`: go to the previous command
- `↓`: go to the next command
- `Ctrl + R <query>`: search in history
- `less .bash_history`: show history
Installing software

Most of the software is available in the system repository. To install a program in Ubuntu type this into terminal:

- `sudo apt update` to update information about available packages
- `sudo apt install <program>` to install the program that you want
- Use `apt search <program>` to find all packages that provide `<program>`
- Same for any library, just with `lib` prefix
We won’t teach you everything about C++

Within C++, there is a much smaller and cleaner language struggling to get out.

-Bjarne Stroustrup
Where to write C++ code

There are two options here:

- **Use a C++ IDE**
  - CLion
  - Qt Creator
  - Eclipse

- **Use a modern text editor** [recommended]
  - Sublime Text 3 [my preference]
  - Visual Studio Code
  - Atom
  - VIM [steep learning curve]
  - Emacs [steep learning curve]

Most icons are from Paper Icon Set: https://snwh.org/paper
Hello World!

Simple C++ program that prints **Hello World!**

```cpp
#include <iostream>

int main() {
    // Is this your first C++ program?
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```
Comments and any whitespace chars are completely ignored

- A comment is text:
  - On one line that follows //
  - Between /* and */

- All of these are valid C++:

```cpp
1 int main() { // Unexpected comment.
    return 0;
}

1 int main()
2 {
3    return 0;
4 }

1 int main() {
    return /* Unexpected comment */ 0;
3 }
```
Good code style is important

Programs are meant to be read by humans and only incidentally for computers to execute.

- Donald Knuth

- Use `clang_format` to format your code
- Use `cpplint` to check the style
- Following a style guide will save you time and make the code more readable
- We use **Google Code Style Sheet**
- Naming and style recommendations will be marked by `GOOGLE-STYLE` tag in slides

Google style sheet: https://google.github.io/styleguide/cppguide.html
Everything starts with main

- **Every** C++ program starts with `main`
- `main` is a function that returns an error code
- Error code 0 means **OK**
- Error code can be any number in \([1, 255]\)

```cpp
1 int main() {
2   return 0;  // Program finished without errors.
3 }
```

```cpp
1 int main() {
2   return 1;  // Program finished with error code 1.
3 }
```
#include directive

Two variants:

- `#include <file>` — system include files
- `#include "file"` — local include files

Copies the content of `file` into the current file

```c
#include "some_file.h"

// We can use contents of file "some_file.h" now.
int main() {
    return 0;
}
```
I/O streams for simple input and output

- Handle `stdin`, `stdout` and `stderr`:
  - `std::cin` — maps to `stdin`
  - `std::cout` — maps to `stdout`
  - `std::cerr` — maps to `stderr`

- `#include <iostream>` to use I/O streams

- Part of C++ standard library

```cpp
#include <iostream>

int main() {
    int some_number;
    std::cin >> some_number;
    std::cout << "number = " << some_number << std::endl;
    std::cerr << "boring error message" << std::endl;
    return 0;
}
```
Compile and run Hello World!

- We understand **text**
- Computer understands **machine code**
- **Compilation** is translation from text to machine code
- **Compilers** we can use on Linux:
  - GCC
  - Clang [*] [used in examples]

Compile and run Hello World example:

```
1  c++ -std=c++11 -o hello_world hello_world.cpp
2  ./hello_world
```
References

- **Cpp Core Guidelines:**
  https://github.com/isocpp/CppCoreGuidelines

- **Google Code Styleguide:**
  https://google.github.io/styleguide/cppguide.html

- **Git guide:**
  http://rogerdudler.github.io/git-guide/

- **C++ Tutorial:**
  http://www.cplusplus.com/doc/tutorial/

- **Book:** *Code Complete 2* by Steve McConnell