UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

INTRODUCTION TO MODERN C++

LECTURE 9

Rémi Géraud

April 7, 2016

École Normale Supérieure de Paris

Lecture 9
Handling large projects.

THIS LECTURE

You now how to create simple C++ projects

- Create source and header files
- Use some libraries
- Compile and run the whole thing

This is good when we work on *small projects*.

LARGE PROJECTS

In a large project, you have

- A lot of source and header files
- Different people with different roles
- A lot of libraries

You don’t want to keep track of all this manually.

LARGE PROJECTS

There are three **essential tools** you need to scale up:

1. A version control system, that keeps tracks of changes and people responsible for them
2. A documentation, that explains what things do what and where to find what and how to use what.
3. An automated build procedure, that takes care of compilation, linking etc.

Today we'll use `git`, `doxygen`, and `make`, respectively.

LARGE PROJECTS

There are also very useful **bonus tools** that are of help:

1. A debugger that helps figuring out where problems come from
2. A profiler that helps finding inefficient code
3. A bug tracker to organise and lead pest control
4. A pile of books to learn and entertain yourselves.

These tools are beyond the scope of today's lecture.

VERSION CONTROL

VERSION CONTROL

Version control solves several **very common problems**:

1. "What was the last version again?"
2. "Who coded *that*?"
3. "Woopsie, I think I messed up. Can I cancel my changes?"
4. "Two people worked on the same code"
5. "My laptop crashed, I lost everything"

VERSION CONTROL

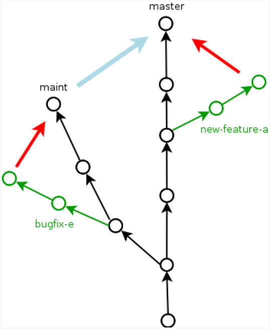
This is achieved by archiving all subsequent versions of a document.

Vocabulary:

- **Update:** Synchronise yourself with the latest version
- **Commit:** Timestamp a new version
- **Merge:** Take two versions of a document and make a third
- **Conflict:** Incompatible versions of a document
- **Branch:** Independent sequence of versions

VERSION CONTROL

This is best visualised by a tree:



E.g. to work on a new feature, you would create a **branch**, implement the new feature, perhaps make several **commits** on the way, and finally **merge** with the main branch, usually called “trunk” or “master”.

VERSION CONTROL

The most common version control systems used today are:

`git` and `svn`

To install them:

```
sudo apt-get install git svn
```

VERSION CONTROL

They use `git` (and so will we) :

- Linux
- VLC
- Facebook
- Microsoft
- nVidia

You can check some projects on `GitHub`.

DOCUMENTATION

DOCUMENTATION

Documentation serves three purposes:

1. Coders: Remember how and why things work
2. Architects: Understand the overall design
3. Users: Know how to use the program

Documentation should be **exhaustive and clear**.

In the end, documentation is what makes the difference between a dying project and a thriving project.

DOCUMENTATION

Use a **standardised documentation format** so that

- Documentation is uniform in content and quality (in spite of many authors)
- Users know where to look for answers (principle of minimal surprise)
- It is easy to have an overview of the whole project at different scales
- Documentation can be automatically generated

DOCUMENTATION

Today we will use `doxygen` and the `JavaDoc` or `QtDoc` documentation format. It automatically turn code annotations into a full-fledged documentation.

To install:

```
sudo apt-get install doxygen
```

But today we'll fetch it from GitHub:

```
https://github.com/doxygen/doxygen
```

and compile and install it ourselves.

DOCUMENTATION

Code annotations look like this (JavaDoc format)

```
/**
* This function finds the answer.
* This is a more elaborate description of this function.
*   @param   myMan The name of the captain
*   @returns       The answer to everything
*/
int FindAnswer(const std::string& myMan) {
  int age;  /**< Age of the captain */
  int size; /**< Size of the boat   */

  // ...

  return 42;
}
```

Note: You *must* document the file (`@file`).

DOCUMENTATION

Then `doxygen` can automatically turn this into documentation.

We can create the configuration with `doxygen -g` or `doxywizard`.

This doesn’t prevent you from providing usable and relevant information.

DOCUMENTATION

They use `doxygen` (and so will we):

- Adobe
- Apache
- Apple
- IBM
- KDE

BUILD MANAGEMENT

BUILD MANAGEMENT

A simple C++ project compilation command may look like

```
g++ vector.cpp matrix.cpp blas.cpp main.cpp -o
program -O3 -fPIC -ffast-math
-fstack-protector-strong -lSDL -lcurl
-D_FORTIFY_SOURCE=1 $(xml2-config ––cflags ––libs)
––std=c++14
```

Now, this gets ugly very fast. Do we *really* have to type the whole thing each time?

BUILD MANAGEMENT

A **build management** system takes care of

- Compiler options
- Source and header file lists
- Libraries and linking options

This is practical for small projects, and *necessary* for medium to large projects.

BUILD MANAGEMENT

We will use `make`, which is the standard build management system on all Unix systems.

Concretely, we will have to write a `Makefile`.

This can be done by hand, but we'll use `autotools` to do it for us.

```
sudo apt-get install automake autotools
```

Questions?

Lab : Practise on a large project!

Start with `https://github.com/alexdantas/sdl2-platformer`