LifeV installation manual on Ubuntu

This tutorial has been tested on ubuntu 12.04 and 12.10

Stable release from github

1. install the following packages
   - `c++` and `fortran` compilers: `gcc`, `g++`, `gfortran`

     ```
     $> sudo apt-get install build-essential gfortran
     ```

   - install basic algebra packages `blas` and `lapack`

     ```
     $> sudo apt-get install libblas-dev liblapack-dev
     ```

   - install the MPI library `openmpi`

     ```
     $> sudo apt-get install openmpi
     ```

   - install git (control version tool), doxygen (to generate the documentation) and cmake (code configuration)

     ```
     $> sudo apt-get install git gitk gitgui doxygen cmake cmake-gui
     ```

2. Create a directory to store the source code of LifeV, the executables and all the additional libraries needed

   ```
   $> mkdir lifev-env  //or another name at your choice
   $> cd lifev-env/
   ```

3. Download the master branch from github

   ```
   $> git clone https://github.com/lifev/lifev.git
   ```

   a directory name can be specified at the end of the command (default: `lifev`)

4. Clone the cmake configuration files inside the lifev directory

   ```
   $> cd lifev/
   $> git clone https://github.com/lifev/cmake.git
   ```

5. Installation scripts can be found in the `tools/install_scripts` folder. Follow the instructions in the `README.md` file (except for the step 1, already done!). The instruction can be visualized at

   https://github.com/lifev/lifev/tree/master/tools/install_scripts

   In this step, you specify installation and building paths and also which libraries must be installed or their path if you want to use system libraries

Remark: the doxygen documentation is available in the folder `doc/api/html`, in the `lifev` build directory:

```
$> xdg-open doc/api/html/index.html
```

or online at

http://cmcsforge.epfl.ch/doxygen/lifev/

Developers version and branches

1. Generate a public key

   ```
   $> ssh-keygen -t rsa
   ```

   This will create two files in `$HOME/.ssh`, a private key `id_rsa` and a public one `id_rsa.pub`

2. Subscribe as a LifeV developer at

   http://cmcsforge.epfl.ch

   There are also mailing lists available (see the CMCSForge site). The previously generated public key is used to authenticate the user in the git repository.

3. Add a new remote repository to the `lifev` and `cmake` git repositories (you must be in a folder tracked by the correspondent git repo)

   ```
   $> git remote add epfl git@cmcsforge.epfl.ch:lifev.git
   $> git remote show epfl // test if the add went well
   $> git fetch epfl
   ```

   The `show` command will display all the available branches on the remote repository.

4. Download a development branch

   ```
   $> git checkout branch_name
   ```

   or create a new one with `git branch`

5. Get the `lifev-config.sh` script from here and edit it setting up proper values for the `LIFEV_SRC_DIR`, `LIFEV_BUILD_DIR` and `LIFEV_INSTALL_DIR`. Use it to compile the source code

   ```
   $> source lifev-config.sh # additional flags here
   ```

   All additional flags will be forwarded to the `cmake` command. Options are in the form

   ```
   -D OPTION_NAME:OPTION_TYPE=VALUE
   ```

   Typical options are

   - `CMAKE_BUILD_TYPE:STRING=Release/Debug` to set up the code in optimized or debug mode
   - `LifeV_ENABLE_Module:BOOL=ON/OFF` to enable or disable a module (the list of modules is in the `PackageList.cmake` file in the source directory)
   - `LifeV_ENABLE_ALL_PACKAGES:BOOL=ON/OFF` to enable or disable all modules at once
   - `LifeV_ENABLE_TESTS:BOOL=ON/OFF` to enable or disable the tests for all the modules
   - `Module_ENABLE_TESTS:BOOL=ON/OFF` to enable or disable the test for a single module

- `CMAKE_CXX_FLAGS:STRING="-O3 -mtune -march"` to set up compilation flags
- `LibraryName_INCLUDE_DIRS:PATH=/path/to/libname/include` and `LibraryName_LIBRARY_DIRS:PATH=/path/to/libname/lib` to set up an external library that is not installed in the system paths

6. Go to the build folder and compile the code

```
$> make # use -jN to speed up compilation using N parallel builds
```

7. Optionally run the testsuite to unsure that everything went ok

```
$> ctest
```