# PLearn Installation Guide

How to install the PLearn Machine-Learning library and tools

August 3, 2022

# Contents

# Chapter 1

# Overview of requirements

**Note:** most of the tools and libraries required for PLearn are already installed on typical Linux (or other unix-like) systems, or are easy to install with ready-made packages (such as RPMs or using apt-get). PLearn is mostly developped in a Linux environment with the gcc/g++ C++ compiler. Since PLearn was written with portability in mind, it is certainly portable (and has been ported on occasions) to other platforms or compilers, but as we work mainly with Linux/gcc, we only very occasionally check the working and correct problems on other platforms.

## 1.1 Required tools

To be able to download, compile and use PLearn, you need the following tools to be installed on your system (for detailed instructions for installation under Windows, you may go directly to section 3.3).

- **g++** (`http://gcc.gnu.org`). We recommend using the latest version, but it should work with 3.2 and above. It is certainly possible to compile PLearn with other compilers (we do this from time to time), but it may necessitate some tweaking.
- **python** (`http://www.python.org`). Version 2.3.3 or above (may work with older versions, but no guarantee). We use python scripts heavily to make our life easier, noticeably as our compilation framework (but if you're really *in love* with Makefiles you can probably do without pymake).
- **subversion (svn)** for version control ( see `http://subversion.tigris.org/` ) Note: some local subprojects may still be based on CVS for version control (`http://www.cvshome.org`)

## 1.2 External library dependencies

Parts of PLearn depend on the following external libraries. Depending on which parts of PLearn you want to compile and use, they may require compiling and linking with the following libraries (that will first have to be installed: see more detailed installation instructions for your system in the appropriate chapter).

- **required:** the standard C and C++ libraries, naturally!
- **required: NSPR** the Netscape Portable Runtime (http://www.mozilla.org/projects/nspr/). We use NSPR as an OS abstraction layer for things such as file and network access, process control, etc... So that PLearn may also somewhat work on Windows...
- **required:** parts of PLearn have come to depend on some of the **boost** C++ libraries (http://www.boost.org). You might as well install all of boost. We use in particular: *tuple, random, type_traits, call_traits, smart_ptr, bind, function, graph, (utility), (regex), (format), (thread).*
- **almost required: blas** and **lapack** libraries provide linear algebra subroutines that several learning algorithms depend upon. There are standard packages (RPMS…) for most Linux distributions, and come as part of the *veclib* framework under Mac OS X.
- **recommended: ncurses** is used in some places for text mode gui.
- **optional: python runtime** is needed for embedding python in your plearn executable if you need them to evaluate python code snippets.
- **optional:** PLearn has an interface to call the **Torch** library, which offers a number of additional learning algorithms (in particular SVMs).
- **optional:** The libraries of **WordNet** for work on language models.
- **optional: MPI** libraries for parallelization.

## 1.3 Python related dependencies

Nearly all higher level GUI demos and plotting facilities are written in python (which calls upon plearn) using essentially the following software packages:

- **numpy** (part of scipy) for efficient numeric array operations in python.

- **matplotlib** for 2D plots.
- **mayavi** for 3D interactive plots.
- **pygtk** with **gtk+2** for sophisticated GUIs.
- **tkinter** for older and simpler GUIs.

You may also want to install the more user-friedly python command-line interface **ipython**: `ipython --pylab` starts a good environment for scientific computing in python.

### 1.3.1 Other useful external tools

In addition the following tools can be useful:

- **gs** (ghostscript) and **gv** (ghostview)
- **gnuplot** (for older plot commands called from plearn)
- **perl** to run older dirty scripts.
- **graphviz** to plot graphs.

## Chapter 2

# Downloading PLearn

After having been hosted for several years on SourceForge (under CVS), as of 2005/06/21 PLearn development has been moved to BerliOS (`http://developer.berlios.de`) that offered the benefits of a faster hosting with the more modern Subversion (SVN) version control system. The latest version is always available through SVN access.

## 2.1 Anonymous SVN checkout

If you just want to hack PLearn locally, you can do an anonymous checkout (no need for a BerliOS account) with

```
svn checkout svn://svn.berlios.de/plearn/trunk PLearn
```

## 2.2 Developer SVN access

If you are going to be a serious contributor to PLearn, you should create a BerliOS account for yourself, and ask to be added to the developer list. This will give you read/write access to the PLearn repository.

Make sure you first move any older version of PLearn out of the way, for ex. by renaming it PLearn.old You can then check-out a fresh copy of PLearn with the following instruction:

```
svn checkout svn+ssh://USERNAME@svn.berlios.de/svnroot/repos/plearn/trunk PLearn
```

where you replace ”USERNAME” by your Berlios username. You will be asked for your Berlios password twice.

If you don’t want to be bothered with svn asking passwords, clisk on *Account Maintenance* on the left panel of your BerliOS personal page, and on *Edit keys* on the bottom of the Maintenance page. You can copy your ssh public key there. (Note: your ssh public key is normally found in your /.ssh/*.pub If it’s not there, you can do a ssh-keygen). As for many changes within BerliOS, it may take a while before this is propagated and taken into account.

We also suggest that you edit your `~/.subversion/config` and look for the line containing `global-ignores`: uncomment it and add OBJS in the list of ignored patterns, to avoid being annoyed with these directories when doing a svn status command. You’ll also have to uncomment the line `[miscellany]` three lines above.

## 2.3 SVN basics

From within your local PLearn directory:

- `svn update` will update your local version with the latest version in the repository.
- `svn commit` will commit your changes to the repository.
- `svn add` will add to the repository the files and directories you pass as argument, **recursively**, so make sure you *really* want to add all those directories’ *full* content…
- There is also `svn rm, svn mv,` and `svn cp` to reorganize the files.

Unlike CVS, most subversion commands are recursive by default. Check the help for a particular command before using it if you are unsure.

For more details, there is a excellent free subversion book online available at: http://svnbook.red-bean.com/en/1.1/index.html

If you don’t have the time to at least peruse the whole book, I would still strongly recommend that you at least read appendix A: Subversion for CVS users: http://svnbook.red-bean.com/en/1.1/apa.html

## 2.4 Overview of the directory structure

Your checked out PLearn has the following high level directory structure:

- `scripts/`
  contains mostly python and perl scripts

  - `pymake` is our build system
  - `pytest` implements our test suite framework
  - `pyskeleton` is our "wizard" or "template" system; it uses source code templates in the `Skeleton/` directory
  - `pypoints` allows graphical interactive input of points (for 2D classif or 1D regression)
  - `pyplot` allows plotting learner outputs (classif decision surface, density plot, etc...)
  - `xpdir` lists the content of PLearn experiment directories
  - `perlgrep`, `search-replace`, and `undo-search-replace` allow for simple code lookup and transformations.
  - `pytansform` and the transformations in `Transformations/` allow for complex code transformations.
  - `cvschangeroot` is used to recursively change the locally recorded cvs root. This is useful to indicate that the repository has moved. Note however that we now use a SVN repository for PLearn rather than CVS.
  - ...

- `commands/` contains the source code (as `.cc` files) of PLearn ”executables” to be compiled with `pymake`. If you look at the `plearn_*.cc` files closely, you will notice that they are all built in the same manner, simply including a number of things they need, and invoking a single function `plearn_main`. So they only differ in the functionalities they #include. They have been arranged according to the external library dependencies that will be needed to compile and link each of them. In short:

  - `plearn_noblas` depends only on NSPR, boost (if you don't have blas, it must be compiled with `pymake -noblas plearn_noblas` )
  - `plearn_lapack` depends on NSPR, boost, BLAS, LAPACK
  - `plearn_curses` depends on NSPR, boost, BLAS, LAPACK, ncurses
  - `plearn_python` depends on NSPR, boost, BLAS, LAPACK, ncurses, python runtime libraries

  Since `plearn_noblas` has the smallest number of requirements, it should be the easiest to get to compile and link. But several important learning algorithms require LAPACK. `plearn_lapack` will contain the most useful 99% of PLearn.

- commands/PLearnCommands/ contains the source code for all PLearn *commands*. These commands are included in the above plearn_* programs and can be invoked with these programs in a command-line fashion.
- commands/language/ contains some programs for manipulating language corpus and WordNet related stuff.
- python_modules/
  The root for python module namespace. Must be in your PYTHONPATH.
  It contains mostly a plearn/ subdirectory, which allows to import plearn.foobar from python.
- doc/
  directory within which Pearn's documentation is generated
- examples/
  contains examples of PLearn scripts
- test_suite
  contains part of the PLearn test suite
- plearn/
  contains all of PLearn's base C++ classes organised in themed subdirectories.
- plearn_learners/
  contains all of PLearn's C++ learner classes organised in themed subdirectories.
- plearn_learners_experimental/
  may contain some *very experimental* C++ learner classes...
- plearn_torch/
  contains C++ classes to interface with the Torch library ( http://www.torch.ch/ )
- pylearn/
  is the beginning of a BoostPython interface to access PLearn from python, but is currently not the way we call upon PLearn from python.

The plearn* directories contain C++ source code (.h and .cc), and as your root PLearn directory is in the -I directive of the compilation commands, this allows to include relevant files by directives such as, for ex:

```
#include<plearn/base/Object.h>
#include<plearn/io/PStream.h>
#include<plearn_learners/generic/PLearner.h>
```

# Chapter 3

# Installing PLearn

The installation of PLearn consist of three phases: installing the dependensys of PLearn, configuring the environnement and we finish with the compilation of PLearn. The compilation is done with pymake witch use python. To have more information on pymake do: `pymake help`.

The next subsection discuss some other information about pymake that is cross-platform. After that their is sections that discuss the installation on linux, on Mac OS X and on windows with cygwin.

### 3.0.1 Cross-platform information about pymake

To have more information on pymake do: `pymake help`

To clean all the file generated during the compilation do `pymake -clean [dir]`. The dir parameter is optinal and if it is not there, the current directory will be used.

(Not sure this is cross-platform, tested on linux) If PLearn is on NFS(or other non local directory), you can speed up the recompilation+liniking, with the `-tmp` or the `-local_ofiles` options. Both will put the objects files in the local directory `/tmp/.pymake` instead of in the PLearn directory. This can considerably speed up the linking phase. The `-tmp` options will compile all the objects files only with the local host. When it is executed for the first time on a computer, it will compile all files. The next time, it will not recompile the previously compiled files if not needed. This is usefull is you have limited space on the PLearn directory on NFS, as there won't be any objects file in it, but if you move to another local computer, you must recompile everything.

The `-local_ofiles` option will distribute the compilation on many computer(see next paragraphe). When executed the first time, it will copy the objects files from the PLearn directory to the `/tmp/.pymake` directory. Then it will recompiles modified files and then link them in this directory. Finaly it will copy them in the PLearn directory. This way, if you change of computer, you won't need to recompile everything, but it will need more space in the PLearn directory. So the only advantage of `-tmp` over `-local_ofiles` is that it take less space in the directory of PLearn, but both will link at the same speed.

`pymake` support the compilation on multiple computer for faster compilation. The list of host is in a file in the directory /.pymake/. To know the name of the file run the compile command wanted. It will give you a line that look like this: `(create a linux-i386.hosts file in your .pymake directory` In this exemple, the file is `linux-i386.hosts`. On this file, you must put one host by line and those hosts most be of the same architecture that the one who start the compilation. If you want the computer that start the compilation to participate in the compiling, it must be included in the file.

The default compilation mode is in debug mode (-dbg). To use other mode, add it as a paramater to the compilation line like this: `pymake -opt plearn_curses.cc`. Here is the list of compilation mode:

- -dbg: debug mode (default)
- -opt: optimized
- -pintel: parallelized for intel compiler
- -safeopt: safe optimized mode (includes bound checking)
- -safeoptdbg: safe optimized mode (includes bound checking), w/ debug info
- -checkopt: some variation on optimized mode
- -gprof: optimized mode with profiler support (-pg)
- -optdbggprof: optimized mode with profiler support WITH DEBUGGING (-pg)
- -safegprof: safe optimized mode with profiler support (-pg)',
- -genericvc++: 'Generic compilation options for Visual C++: the debug/opt options are actually set directly in the .vcproj project file',

## 3.1 Installation on Linux

### 3.1.1 PLearn setup and compilation

We suppose you have all the necessary software requirements in place (especially python).

For ex. under ubuntu, you can check the following packages install (using synaptic for ex.):

Required: `g++, python2.4, python, libnspr4, libnspr-dev, libboost-dev, libboost-regex1.33.1, libboost-regex-dev, libboost-graph1.33.1, libboost-graph-dev`
Strongly suggested: `libncurses5, libncurses5-dev` and some version of lapack and blas. Ex: `refblas3, lapack3, lapack3-dev`
Recommended (python and graphics related): `ipython, python-matplotlib, python-numarray, python-numpy, mayavi, python2.4-dev, python-tk, python-gtk2, python-gtkhtml2`

You can use the following command under an Ubuntu (probably other derivative of Debian) to install the required packages: `sudo apt-get install g++ python2.4 python libnspr4 libnspr-dev l`
The following command for the strongly suggested packages: `sudo apt-get install libncurses5 libncurs`
And the following command for the recommanded packages:`sudo apt-get install ipython python-matplot`
The package that are alreay installed won't be reinstalled.

Then edit your .cshrc or .bashrc and

- Set the `PLEARNDIR` environment variable to the path of your PLearn directory.
  - csh ex: `setenv PLEARNDIR ${HOME}/PLearn`
  - bash ex: `export PLEARNDIR=${HOME}/PLearn`
- Append `$PLEARNDIR/scripts` and `$PLEARNDIR/commands` to your path.
- Append `$PLEARNDIR/python_modules` to your `PYTHONPATH`
  - csh ex: `setenv PYTHONPATH $PLEARNDIR/python_modules`
  - bash ex: `export PYTHONPATH=$PLEARNDIR/python_modules`
- Restart your shell for these changes to take effect.

From within your PLearn directory run `./setup`. This should create a `.plearn` sub-directory in your home directory, that will contain some configuration files.

Now you should be able to try and compile a first version of plearn. We have our own make system based on a python script (`pymake`) that automatically parses source files for dependencies and determines what to compile, and what to link (including optional libraries), and is able to run parallel compilation on several machines. It is easily customizable.

The compiling commands for the version with the minimum number of dependencies are:

```
cd PLearn/commands/
pymake -noblas plearn_noblas.cc
```

If it doesn't work, you may have to adapt the configuration file to your system (PLearn/.pymake/config)

If it does work, you can try with more dependencies to have more fonctionality with the commands `pymake plearn_lapack.cc`, `pymake plearn\_curses.cc` or even `pymake plearn_python.cc`

## 3.2 Installation on Mac OS X

### 3.2.1 External dependencies (Mac OS X 10.5 "Leopard")

The easiest way to install external dependencies is through fink. You should install **fink** ( `http://fink.sourceforge.net` ) And its GUI **Fink Commander** ( `http://finkcommander.sourceforge.net/` ). To gain access to the most up-to-date packages, enable the *use of unstable package* (e.g. in `Menu Fink Commander / Preferences.../ Fink` ). To be able to compile and link the core of plearn, you should install the following packages through fink:

- `python25`
- `boost1.33`
- `nspr` and `nspr-shlibs` (for NSPR)
- `scipy-core-py25` (for numpy, which somehow creeped in the dependencies of pymake, and shouldn't have...)
- `ncurses` (useful for viewing dataset tables)

Optional libraries also easily installable from fink:

- `matplotlib-py25` for 2D graphics
- `mayavi-py25` for 3D interactive graphics
- `pygtk2-py25` if you want to play with the python GUI tools and demo.

### 3.2.2 External dependencies (older version of Mac OS X)

If you want to see graphical displays, you should also install X11 (apple has a version on its system install CDS shipped with the computers as part of XCode: look for packages **X11 User** and **X11 SDK**.

You should then install **fink** ( `http://fink.sourceforge.net` ) And its GUI **Fink Commander** ( `http://finkcommander.sourceforge.net/` ). We recommend you also enable the *use of unstable package* (in `Menu Fink Commander / Preferences.../ Fink` ) to gain access to the latest packages.

To be able to compile and link the core of plearn, you should install the following packages through fink:

- `python23`
- `boost1.32-py23`
- `mozilla-dev` (for NSPR)

Optional libraries also easily installable from fink:

- `ncurses` (useful for viewing dataset tables)
- `fpconst-py23`
- `numarray-py23` for efficient matrix/vector manipulations in python
- `scipy-py23` for efficient matrix/vector manipulations in python
- `matplotlib-py23` for 2D graphics
- `mayavi-py23` for 3D interactive graphics
- `pygtk2-py23` if you want to play with the python GUI tools and demo.

### 3.2.3 Environment setup

You should make sure the following variables and paths are correctly defined in your environment variables (in your `.profile` with your `.bashrc` simply doing a `source $HOME/.profile`)

```
# The default fink installs its packages in /sw and should already have
# added the following line.
source /sw/bin/init.sh

# adapt to your configuration if your PLearn directory is not $HOME/PLearn
export PLEARNDIR=$HOME/PLearn
export PATH=/sw/bin:/sw/sbin:$PLEARNDIR/scripts:$PLEARNDIR/commands:.:$PATH
export LD_LIBRARY_PATH="/sw/lib:/sw/lib/mozilla:$LD_LIBRARY_PATH"
export LIBRARY_PATH=$LD_LIBRARY_PATH
export CPATH="/sw/include:/sw/include/mozilla:$CPATH"
export PYTHONPATH="$PLEARNDIR/python_modules:$PLEARNDIR/scripts:$PYTHONPATH"
export SKELETONS_PATH=$PLEARNDIR/scripts/Skeletons
```

Restart a new shell for these to take effect. Make sure they're well defined in the new shell.

### 3.2.4 PLearn setup and compilation

From within your PLearn directory run `./setup`. This should create a `.plearn` sub-directory in your home directory, that will contain some configuration files. The compiling commands for the version with the minimum number of dependencies are:

```
cd PLearn/commands/
pymake -noblas plearn_noblas.cc
```

If it doesn't work, you may have to adapt the configuration file to your system (PLearn/.pymake/config)

You can try with more dependencies to have more fonctionality with the commands `pymake plearn_lapack.cc`, `pymake plearn_curses.cc` or even `pymake plearn_python.cc` or `pymake plearn_full.cc`

To compile the plearn python extension module, do `make_plearn_python_ext`

## 3.3 Installation on Windows with cygwin

This section describes a step-by-step installation under the Microsoft Windows environment. Note that the following instructions are outdated.

Cygwin (`http://cygwin.com`) is a Linux-like environment for Windows, and is currently the easiest route to using PLearn under Windows.

### 3.3.1 Installing Cygwin

Download from `http://cygwin.com/setup.exe` the latest Cygwin setup program, then run it. Select your installation options (you should keep the recommended Unix / binary default text file type). Once you reach the "Select Packages" step, click the "View" button to switch to full view and select the following packages to install (the version number in parenthesis was the version used when this guide was written, hopefully any further version should work too).

- `autoconf` (2.59-2)

- gcc-g++ (3.4.4-1)
- gcc-g77 (3.4.4-1)
- lapack (3.0-3)
- libncurses-devel (5.4-4)
- make (3.80-1)
- python (2.4.1-1)
- subversion (1.2.3-1)

Optional (but recommended) packages to install:

- gdb (20041228-3)
- tcsh (6.14.00-5)
- unzip (5.50-5)
- vim (6.4-2)

The default shell installed with Cygwin is bash, but in the following, we will be using tcsh (though you may of course adapt the instructions below to get PLearn to work with bash). Assuming you have installed the tcsh package and Cygwin is installed in `C:\cygwin`, edit `C:\cygwin\cygwin.bat` and replace the line `bash --login -i` with `tcsh -l`. You may also change the default location of your home directory by adding the following line at the beginning of the file (make sure you do not have blanks in the path you provide):

```
@SET HOME=C:\MyCygwinHome
```

Note that it is suggested to use a "low-level" path for your home directory, i.e. as close to the root as possible, because of the limitations of Windows concerning the length of paths (which may cause some tests to fail in the test-suite).

### 3.3.2 Installing Boost

The next step is to install the Boost library. You could install part of it from the Cygwin setup utility, but you would neeed to compile Boost-Python anyway. Go to `http://sourceforge.net/projects/boost/` then to the "Files" section to download the Boost library. Download the `tar.gz` files: you will need the source for both Boost and Boost-Jam (the Boost installer). Move these files to your Cygwin home directory. The files used when writing this guide were `boost_1_33_1.tar.gz` and `boost-jam-3.1.11.tgz`.

You may now run Cygwin and launch the Boost installation:

1. 
   ```
   tar zxvf boost_1_33_1.tar.gz
   ```
2. 
   ```
   tar zxvf boost-jam-3.1.11.tgz
   ```
3. 
   ```
   cd boost-jam-3.1.11
   ```
4. 
   ```
   sh ./build.sh
   ```
5. 
   ```
   cd ../boost_1_33_1
   ```
6. 
   ```
   ../boost-jam-3.1.11/bin.cygwinx86/bjam.exe -sTOOLS=gcc
   --prefix=$HOME/local -sPYTHON_ROOT=/usr
    -sPYTHON_VERSION=2.4 --builddir=/tmp/boost_build install
   ```

The installation command above will install Boost in the `local` directory under your home directory: you may remove the `--prefix` option to perform a system-wide installation (however, installing under your home directory is often necessary for Linux users without administrator rights). Do not worry about some of the Boost libraries not compiling, PLearn does not need all of them. Let us just create the appropriate links for those needed by PLearn:

1. 
   ```
   cd ~/local/lib
   ```
2. 
   ```
   ln -s libboost_regex-gcc-mt-1_33_1.dll libboost_regex.dll
   ```
3. 
   ```
   ln -s libboost_python-gcc-mt-1_33_1.dll libboost_python.dll
   ```
4. 
   ```
   cd ~/local/include
   ```
5. 
   ```
   ln -s boost-1_33_1/boost
   ```

### Installing NumArray

Download NumArray from
`http://www.stsci.edu/resources/software_hardware/numarray`
then install it with the following commands:

```
tar zxvf numarray-1.5.0.tar.gz
cd numarray-1.5.0
python setup.py config install --gencode --prefix=$HOME/local
```

Note that (at least in version 1.5.0) there is a typo in a NumArray C++ file. After you have installed NumArray with the commands above, edit `~/local/include/python2.4/numarray/arraybase.h` to manually remove the comma at the end of line 117. Additionally, if you want to get rid of a gcc warning when compiling PLearn, you can edit `libnumarray.h` in the same directory and replace line 51 with

```
static void **libnumarray_API __attribute__ ((unused)) ;
```

### 3.3.3 Installing NSPR

Go to `ftp://ftp.mozilla.org/pub/mozilla.org/nspr/releases` to get the NSPR release for your operating system (in this guide we used the 4.6 release for Windows NT 5.0, in optimized mode). Extract the zip file to a temporary directory, then move the content of the `include` directory (including its sub-directories) to `~/local/include/mozilla/nspr` (that you will need to create), and the `libnspr4.dll` file (in the `lib` directory) to `~/local/lib`. Check the permissions for `libnspr4.dll`: you need the "execute" permission for PLearn to be able to run. You can set it with:

```
chmod u+x ~/local/lib/libnspr4.dll
```

### 3.3.4 Environment setup

First, go to the directory where you wish to install PLearn (in this guide we will assume this is your home directory), and check out the latest version from the Subversion repository:

```
svn checkout svn://svn.berlios.de/plearn/trunk PLearn
```

Since `pymake` will create directories named `OBJS` to store compiled object files, Subversion should ignore them: edit your `~/.subversion/config` and, in the `miscellany` section, write the line

```
global-ignores = OBJS
```

Now, edit your `~/.cshrc` and add the following lines:

```
# Environment variables.
setenv PLEARNDIR ${HOME}/PLearn
setenv PATH /usr/local/bin:/usr/bin:/bin:/usr/lib/lapack:
```

```
            ${PLEARNDIR}/commands:${PLEARNDIR}/scripts:
            ${HOME}/local/lib
setenv PYTHONPATH ${PLEARNDIR}/python_modules:
                  ${HOME}/local/lib/python2.4/site-packages
setenv CPATH ${HOME}/local/include
setenv LD_LIBRARY_PATH ${HOME}/local/lib
setenv LIBRARY_PATH ${LD_LIBRARY_PATH}

# Nicer prompt.
set prompt = "%B%m %~%b > "
```

You need to redefine the `PATH` environment variable because the original one will usually contain directories with blanks (such as `Program Files`), which Cygwin has trouble with. The last line is very optional (it just gives you a nicer prompt). Now edit `$PLEARNDIR/pymake.config.model` and look for the `python` optional library. Just before, add the following lines:

```
python_version = '2.4'
compileflags += ' -DPL_PYTHON_VERSION=240'
python_lib_root = '/usr/lib'
numpy_site_packages = join(homedir,
             'local/lib/python2.4/site-packages/numarray')
```

Do a `source ~/.cshrc` to reload your configuration file, then go to your PLearn installation directory (`$PLEARNDIR`) and run `./setup`. PLearn can now be compiled with `pymake $PLEARNDIR/commands/pl`

# Chapter 4

# Other installation-related information

## 4.1 Testing PLearn

Once PLearn compiles successfully, it is recommended to run the test-suite in order to ensure that the main components of the library work as expected. Go to `$PLEARNDIR` and launch `pytest run --all`. Note that, typically, the test-suite first compiles `$PLEARNDIR/commands/plearn_tests`, and thus may look like it is stalled, while it is actually compiling or linking in the background.

## 4.2 Generating the documentation locally

The `doc` subdirectory maintains the sources of the documentation that is also available on the PLearn website. To generate it locally from source you'll need the following software tools:

- A working LaTeX distribution such as `teTeX`
- `latex2html`
- `doxygen`

# License

This document is covered by the license appearing after the title page.

The PLearn software library and tools described in this document are distributed under the following BSD-type license:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.