Instructions for Installing Albany/LCM and Trilinos on Fedora 21

The Albany/LCM team*

April 15, 2015

1 Introduction

This document describes the necessary steps to install Albany/LCM and Trilinos on a machine with Fedora Linux. The procedures described herein were tested using Fedora 21.

If you want a shortcut obtain the script `install_albany.sh` (stored in `Albany/doc/LCM/build`) and then try:

```
./install_albany
```

This will install and build Trilinos and Albany in the current directory. If the script does not complete it will tell you why and with help from this document you will be able to complete the install.

2 Required packages

After installing Fedora, the following packages should be installed using the `yum` command:

```
blas
blas-devel
lapack
lapack-devel
openmpi
openmpi-devel
netcdf
netcdf-devel
netcdf-static
netcdf-openmpi
netcdf-openmpi-devel
hdf5
hdf5-devel
hdf5-static
hdf5-openmpi
hdf5-openmpi-devel
boost
boost-devel
boost-static
boost-openmpi
boost-openmpi-devel
matio
matio-devel
libX11
libX11-devel
cmake
gcc-c++
git
```

For example, to install the first package you should type:

*Original version by Julián Rímoli

```
sudo yum install blas
```

Make sure that all these packages are installed, specially if you create a script to do so. If a package is not installed because of a typo then the compilation will fail.

Optional but strongly recommended packages:

```
clang
clang-devel
gitk
```

3 Git repository setup with Github

In a web browser go to `www.github.com`, create an account and set up ssh public keys. If you require push privilges for Albany, email Glen Hansen at `gahanse@sandia.gov` and let him know that. On the other hand, if you require push privileges for Trilinos, it is best if you contact the Trilinos developers directly. Go to `www.trilinos.org` for more information.

It is strongly recommended that you join the `AlbanyLCM` Google group to receive commit notices. Go to `groups.google.com/forum/#!forum/albanylcm` and join the group. You can also browse the source code at `github.com/gahansen/Albany`.

4 Directory structure

In your home directory, create a directory with the name `LCM`:

```
mkdir LCM
```

Change directory to the newly created one:

```
cd LCM
```

Now, check out the latest version of Trilinos. If you have an account on `software.sandia.gov`, then use:

```
git clone software.sandia.gov:/space/git/Trilinos Trilinos
```

Otherwise clone with:

```
git clone git@github.com:trilinos/trilinos.git Trilinos
```

This last public Trilinos repository is a few days behind the one at `software.sandia.gov` which for most purposes will not be an issue.

Finally, check out the latest version of Albany:

```
git clone git@github.com:gahansen/Albany.git Albany
```

At this point, the directory structure should look like this:

```
LCM
  |- Albany
  |- Trilinos
```

5 Installation scripts

Create symbolic links to the installation scripts inside the directory `LCM/Albany/doc/LCM/build` to the top-level `LCM` directory. If you intend to modify these scripts, it is better to copy them. The necessary scripts are:

```
albany-config.sh
build-all.sh
build.sh
env-all.sh
env-single.sh
trilinos-config.sh
```

Once this is done, go to the top-level `LCM` directory, open the `env-all.sh` and `env-single.sh` scripts and make sure they match your environment. If you do not want to tinker with any of this, just change the email addresses for tests reports at the end of `env-single.sh` and make sure all the scripts are executable and read only:

```
cd ~/LCM
chmod +x *.sh
chmod -r *.sh
```

The `build.sh` and `build-all.sh` scripts perform different actions according to the name with which they are invoked. See `LCM/Albany/doc/LCM/build/README` for more details.

For example, the following symbolic links will create separate commands for clean up, configuring and testing:

```
ln -s build.sh clean.sh
ln -s build.sh config.sh
ln -s build.sh test.sh
```

They could also be combined for convenience:

```
ln -s build.sh clean-config.sh
ln -s build.sh clean-config-build.sh
ln -s build.sh clean-config-build-test.sh
ln -s build.sh config-build.sh
ln -s build.sh config-build-test.sh
```

There is also a script `LCM/Albany/doc/LCM/install/albany-lcm-symlinks.sh` that will create the appropriate symbolic links.

6 Environment variables

The following additions to the command shell environment variables are required for proper Albany compilation and execution in this setup:

```
export PATH=/usr/lib64/openmpi/bin:$PATH
export LD_LIBRARY_PATH=/usr/lib64/openmpi/lib:$LD_LIBRARY_PATH
```

7 Configuring and compiling

First, configure and compile Trilinos. Within the top-level `LCM` directory type:

```
./config-build.sh trilinos [toolchain] [build_type] [#_processors]
```

For example, if you want to use the `GCC` toolchain to build a release version of the code using 16 processors, type:

```
./config-build.sh trilinos gcc release 16
```

Finally, repeat the procedure for Albany:

```
./config-build.sh albany [toolchain] [build_type] [#_processors]
```

For example, if you want to use the `GCC` toolchain to build a release version of the code using 16 processors, type:

```
./config-build.sh albany gcc release 16
```

Note that to compile a version of Albany with a specific toolchain and build type, the corresponding version of Trilinos must exist.

Currently the options for the toolchain are `gcc`, `clang` and `intel` if the Intel compilers are installed, and for build type are `debug` and `release`. The `clang` toolchain requires installation of the `clang` and `clang-devel` packages.

8 After initial setup

The procedure described before configures and compiles the code. From now on, configuration is no longer required so you can rebuild the code after any modification by simply using the `build.sh` script. For example:

```
./build.sh albany gcc release 16
```

There are times when it is necessary to reconfigure, for example when adding or deleting files under the `LCM/Albany/src/LCM` directory. This is generally anounced in the commit notices.

Also, note that both Trilinos and Albany are heavily templetized C++ codes. Building the debug version of Albany requires large amounts of memory because the huge size of the symbolic information required for debugging. Thus, if the compiling procedure stalls, try reducing the number of processors.

9 Running and debugging LCM

After building Albany, you might want to run and/or debug the code. Tools were built in Trilinos (decomp, epu, etc.) that are necessary for parallel execution. Please modify your path

```
export PATH=$HOME/LCM/trilinos-install-gcc-release/bin:$PATH
```

and include the necessary libraries

```
export LD_LIBRARY_PATH=$HOME/LCM/trilinos-install-gcc-release/lib:$LD_LIBRARY_PATH
```

prior to execution. Please note that in this specific case, we are pointing to `gcc-release`. If we attempt to execute the debug version, the `LD_LIBRARY_PATH` will be invalid. One is required to change the `LD_LIBRARY_PATH` for each toolchain and build type. To select toolchains and build types for execution, one can create configuration scripts in his/her LCM directory. Each configuration script might be named something akin to `LCM_gcc_release.conf` and contain the necessary changes to `PATH` and `LD_LIBRARY_PATH`. For example, prior to debugging the code one would construct the configuration file `$HOME/LCM/LCM_gcc_debug.conf`

```
export PATH=$HOME/LCM/trilinos-install-gcc-debug/bin:$PATH
export LD_LIBRARY_PATH=/$HOME/LCM/trilinos-install-gcc-debug/lib:$LD_LIBRARY_PATH
```

and then `source $HOME/LCM/LCM_gcc_debug.conf` prior to execution. We note that these environmental variables are temporarily set by the build scripts. However, one may have a total of six toolchains and build types. Currently, the user must specify the environmental variables necessary for execution.

10 Policies

Albany is a simulation code for researchers by researchers. As such, vibrant development of new and exciting capabilities is strongly encouraged. For these reasons, don't be afraid to commit changes to the master git repository. We only ask that you don't break compilation or testing. So please make sure that the tests pass before you commit changes.

In addition, within LCM we strongly encourage you to follow the C++ Google style guide that can be found at `http://google-styleguide.googlecode.com/svn/trunk/cppguide.html`. This style is somewhat different to what is currently used in the rest of Albany, but we believe that the Google style helps the developer more in that it advocates more style differentiation between the different syntactic elements of C++. This in turn makes reading code easier and helps to avoid coding errors.