Instructions for Installing Albany/LCM and Trilinos on Fedora 20

The Albany/LCM team*

October 29, 2014

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 20.

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
cmake
gcc-c++
git
```

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

```
sudo yum install blas
```

Make sure that all these packages are installed, specially of 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:

*Original version by Julián Rímoli

```
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:

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

Finally, check out the latest version of Albany:

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

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

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

5 Installation scripts

Copy the installation scripts inside the directory `LCM/Albany/doc/LCM/build` to the top-level `LCM` directory. The necessary scripts are:

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

Once copied, 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
```

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` and `clang`, 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 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.