LAMPS 90

TUTORIAL

Last update: 17-Jul-2019

Summary:

This tutorial guides the user through the basic steps required to run the LAMPS 90 package. For more complete information on LAMPS 90, see the LAMPS 90 User's Guide.

Author:

Robert Cohen (East Stroudsburg University)

Notice:

LAMPS 90 is written by and for the Numerical Model Development Facility at Drexel University. For further information contact Dr. Robert Cohen at East Stroudsburg University.

Table of Contents

Tutorial 0
Preparing your computer

The LAMPS 90 package is only configured for UNIX machines. To run LAMPS90 you need the unix operating system. Previous knowledge of UNIX is not required, but it is recommended.

LAMPS 90 wlll not run on Windows or Mac. However, it can run on Linux.
You only need to follow this tutorial if you have a computer running Windows 10 and want to run LAMPS on the computer using Linux. Otherwise, you can skip to tutorial #1.

Step - By - Step Guide

Step # 1 ...Install Debian Linux

Search online for the instructions.

Step # 2 ...Update the package list (% indicates a unix command line)

```
% sudo apt update
% sudo apt upgrade
```

Step # 3 ...Install the necessary packages

```
% sudo apt install csh gfortran gcc make time libx11-dev xorg-dev libcairo-dev
libbz2-dev
```

Step # 4 ...Download and extract NCAR graphics (NCL)

Instructions for the appropriate download is avialable on the NCAR website https://www.ncl.ncar.edu/Download/linux.shtml. Once you've downloaded the appropriate tar file and uncompressed it, do the following (use the file name of the downloaded file as <tar-file>).

```
% chdir /usr/local
% sudo tar xvf <tar-file>
```

Step # 5 ...Download and install Xserver software

An example is VcXsrv (search online)

Step # 6 ...Edit your ~/.bashrc file

The following lines need to be added to your .bashrc file

```
             export PATH=.:~/bin/usr/local/bin:$PATH
             export NCARG_ROOT=/usr/local
             export DISPLAY=localhost:0.0
```

You are now ready to install LAMPS on your computer. Proceed to tutorial 1.

Tutorial 1
Getting LAMPS 90 onto your computer

*** * * * READ THIS !!! * * * ***

The LAMPS 90 package may already be on your computer. If you are unsure as to whether it is or not (or whether an update is needed), check with someone who knows before you proceed further. If the LAMPS 90 package is already on your computer, find out where the main LAMPS 90 directory is located and then read this tutorial but DO NOT CARRY OUT THE STEPS.

*** * * * READ THIS !!! * * * ***

Installation Instructions

Step # 1 ..... Get the LAMPS 90 package onto your machine.

Step # 2 ..... Configure the package for your machine.

Step - By - Step Guide

What follows is a more detailed, step-by-step guide for accomplishing the above. The LAMPS 90 package is only configured for UNIX machines. Previous knowledge of UNIX is not required, but it is recommended.

The user should use the UNIX commands listed with each step with care. What actually appears on the screen may not be exactly what is written below. Use these commands as a guide.

Step # 1

Step # 1.1 ... Create a new directory

```
% mkdir lamps90
```

Note: You will probably want to place this directory in some place that all users can have access to it.

Step # 1.2 ... Change your working directory to that directory

```
% chdir lamps90
```

Step # 1.3 ... Get LAMPS 90 tar file

Go to http://quantum.esu.edu/physics/lamps/ *and download the compressed tar file containing the LAMPS package.*

Step # 1.4 ... Extract all the LAMPS 90 files from the tar file

```
% uncompress lamps90.tar.gz
% tar xvf lamps90.tar
```

Note: After this step, you will have a sub-directory named "doc". Within this directory is a copy of the LAMPS 90 User's Guide and LAMPS 90 Programmer's Guide (lamps90u.ps and lamps90p.ps). They are postscript files to be printed out on your postscript printer. Also within this directory are two help files : a UNIX help file and a 'vi' visual editor help file. If you are unfamiliar with the UNIX operating system or you would like to know more about the 'vi' visual editor, check out these files.

Step # 2

Step # 2.1 ... Make sure there is a definition file for your machine

```
% ls config/defs/*def
```

If there are no definition files that match your machine, then you need to read the LAMPS 90 User's Guide, chapter 3.

Step # 2.2 ... Edit the appropriate definition file

For example, if you are working on a SUN, you would do the following.

```
% vi config/defs/sun.def
```

Read through the entire contents of the file to make sure that the definitions listed there are appropriate for your machine. If you don't know anything about the computer you are working on, find someone who does and ask them if the definition file has the appropriate settings.

Note: Use ":wq" to quit the visual editor. See the doc/vi.help file for information regarding the vi editor.

Step # 2.3 ... Use Install

```
% Install
```

This command sets in motion the procedures needed to properly setup LAMPS 90 for your particular machine. All it needs to know is the name of the definition file that is appropriate (see step # 2.2).

After that, it will automatically make any changes that are required by your particular machine as long as it can determine what system you are on. If it cannot, it is up to you to make sure that the package is set up properly. See the LAMPS 90 User's Guide for more information.

Note: If Install complains about not finding fsplit, go to your lamps bin directory and compile the fsplit program there (via "make fsplit").

You are now ready to set-up your user directory. Proceed to tutorial 2.

Tutorial 2
Setting up your user directory

This tutorial guides the user through the basic steps required to set up the LAMPS 90 package so that the user can access the LAMPS 90 programs. For more complete information, see the LAMPS 90 User's Guide.

This tutorial assumes that LAMPS 90 has already been installed as in tutorial 1.

The directory in which you placed lamps90.tar (see tutorial 1) is called the main LAMPS 90 directory. This directory (and associated sub-directories) contains all the programs and such needed to run a LAMPS 90 case. One shouldn't do work in this directory, however, because all the users on your computer will also be using the information in this directory and what you do might mess up what others are doing.

To avoid such complications, each user creates his/her very own directory in which do all the work. This directory is called the user directory. Setting up your own user directory is very simple because there is a command created especially for doing this.

*** * * * READ THIS !!! * * * ***
Once you set up your user directory, you don't need to do it ever again unless the LAMPS 90 package gets reinstalled. Thus, tutorial 2 only needs to be carried out once for each user.
*** * * * READ THIS !!! * * * ***

User Set-Up Instructions

You should be located within the main LAMPS 90 directory.

<u>Step # 1</u> ... Run Setup

```
% Setup user
```

Follow the directions. All you need to do is enter the directory that will be used as your personal LAMPS 90 user directory. One possibility is to just use a sub-directory of your home directory named 'lamps'. This should be the default. If it is okay with you, just press return and accept it as the default (this will produce a sub-directory called lamps within your home directory).

<u>Step # 2</u> ... Change the current directory to your user directory

```
% chdir lamps
```

You are now ready to set up a case directory. Proceed to tutorial 3.

Tutorial 3
Setting up a case directory

This tutorial guides the user through the basic steps required to set up a case directory. For more complete information, see the LAMPS 90 User's Guide.

This tutorial assumes that the user directory has already been set up as in tutorial 2.

After completing tutorial 2, you may be wondering why a special command is necessary to create a user directory. The reason is that it is very important that each program knows where all the other programs are. The Setup command not only makes your user directory, it also allows you to use Setup from there.

From your user directory, you can use "Setup case" to create a sub-directory of your user directory. This sub-directory is called a case-dependent directory (or case directory). You will need to create one case directory for each grid size that you will be using. This is because the array sizes within some of the programs depend upon the grid size to be processed.

Within the case directory, Setup will place the necessary makefiles, include files and sub-directories necessary to run the LAMPS 90 programs.

*** * * * READ THIS !!! * * * ***

You only need to set-up a case directory for each different grid size you will be using. If you plan to run different cases of the same grid size, you don't need to set up a different case directory. Thus, tutorial 3 only needs to carried out once for each grid size.

*** * * * READ THIS !!! * * * ***

Set-up Instructions

You should be located within your user directory.

Step # 1 ... Run Setup

```
% Setup case
```

Follow the directions. You will be asked the name of the case sub-directory. The default is 'case'. This will be fine for now. You will also be asked the grid dimensions to be used. The default is a two-dimensional case (i.e., only 1 grid point in the north-south direction - MNY). This will minimize the amount of disk space needed. For the other grid dimensions, just accept the default values.

Step # 2 ... Change the current directory to the case directory

```
% chdir case
```

You are now ready to run the LAMPS 90 programs. Proceed to tutorial 4.

Tutorial 4
Running programs

This tutorial guides the user through the basic steps required to run the LAMPS 90 programs. For more complete information, see the LAMPS 90 User's Guide.

This tutorial assumes that the case directory has already been set up as in tutorial 3.

*** * * * READ THIS !!! * * * ***

Tutorial 4 shows you a tedious way of running programs. A simpler way is described in tutorials 5 and 6. You are urged to follow tutorial 4 anyway so that you can understand exactly what is being done in each step.

*** * * * READ THIS !!! * * * ***

General Steps

For each LAMPS program, the general steps are as follows. Do not attempt to carry these out now. More specific instructions will come after.

1. Create the program

make <*name*>

The "make" command is utilized as part of the LAMPS 90 makefile structure to compile the necessary programs (see next page). This step will create an executable file called bin/<name>.

2. Start the program

bin/<*name*>

By typing the name of the executable file, the program is automatically started.

3. Type in the necessary commands

.<*command*> <*parameters*>

Once the program is started, it awaits further commands from the user. It is up to the user to know what commands are necessary to get the program to do what is desired.

Note: The number sign (#) acts as a comment delimiter. Any characters occuring in a line after the number sign will be ignored by the program.

LAMPS 90 Programs

There are three kinds of LAMPS 90 programs.

```
i.    initialization programs

   create data for input to dynamic model

   i.a.  analysis of observed data
      begin    : set-up initialization terrain, sfc type and sst
      sound    : read in sounding data
      frstg25  : read in NMC 2.5 degree first guess field
      frstg381 : read in NMC 381 km first guess field
      lyrmean  : create layer-mean averages of sounding data
      rhanal   : analyze relative humidity data
      isenv2p2 : analyze winds and met data
      vertran  : translate data to model grid
      wdadjust : adjust winds
      fgsa3d   : read in first guess field for 3-D analysis
      masa3d   : analyze mass field for 3-D analysis
      wina3d   : analyze wind field for 3-D analysis
      reha3d   : rehash fields for model PI levels
      terrain  : create terrain data volume
      merge    : read data volume and create input for model
      gdinterp : interpolate data to a different grid

   i.b.  creation of analytical data
      analytic : analytic initialization

ii.   dynamic model programs

   simulate evolution of atmosphere given initial state

   ii.a. realistic simulations with full physics
      model    : dynamic model
      tbound   : create boundary tendencies from merge output

   ii.b. simplistic simulations with no moisture
      modeldry : simple dry dynamic model

iii.  post-processing programs

   plot results of dynamic model simulation

   iii.a.   cross-section, horizontal, skew-t plots
      plta3d   : create plots from 3-D analysis
      pltmout  : make plots of variables from history
                   (horizontal, x-section, difference and skew-ts)
   iii.b.   trajectories
      trajec   : calculate trajectories from history
      trajplt  : plot trajectories
   iii.c.   other
      dvutil   : access data volume information
      reformat : reformat history data
```

Naturally, the program you use depends upon your purpose. To illustrate how one goes about running LAMPS 90, this tutorial will assume that the user is a novice who is interested in running a simulation using analytic initial data (i.b.), based on a dry, simplistic dynamic model (ii.b). The programs needed to carry this out are as follow.

1. analytic

   - creates a file which contains the initial state of the atmosphere

2. modeldry

   - reads in the initial state (as produced by analytic)
   - simulates the evolution of the atmosphere from the initial state
   - produces a history of the evolution (the history file)

3. pltmout

   - creates graphs from the history file
   - the graphs are stored in a metacode file named 'gmeta'

The metacode file produced by pltmout is readable by various NCAR graphics utilities (like plt, ictrans or idt). These utilities can then make the graphs viewable on the screen or send them to be printed.

Step - By - Step Guide

You should be located within your case directory.

Analytic

Step # 1 ... Create the analytic initialization program

```
% make analytic
```

Note: The first time "make" is invoked, all the necessary library routines will automatically be compiled. Do not be alarmed if this process takes a while to complete.

Step # 2 ... Start the analytic initialization program

```
% bin/analytic
```

Step # 3 ... Type in the commands for the analytic program

```
# You only need to type the characters before the "#" in each line
# LAMPS programs are designed to ignore all characters after the "#"

.help # This command isn't necessary but it will list out all of the
      # valid commands for the analytic program.
```

```
.htop   5500.
      # This specifies the height at which the sigma levels become flat.

.levels
    0.0         25.0       375.0      750.0      1250.0
    2000.0      3000.0     4000.0     5000.0     6000.0
    7000.0      8000.0     9000.0     10000.0    11000.0
    12000.0     13000.0    14000.0    15000.0    16000.0
      # This specifies the height (in meters) above elevation that you
      # want each level of the model to be at.  Format is not important.
      # Just leave a space or carriage return between each entry.  It
      # *is* important, however, to make sure that the number of levels
      # entered here is the same as the number of levels specified
      # during the Setup case step (see tutorial 3).

.grid wlon -100. slat 15. dlat 1.25
      # This specifies the western boundary (in degrees east of
      # Greenwich), the southern boundary (in degrees north) and the
      # grid step (in degrees latitude) for the horizontal grid that you
      # will be using.

.file init.dat
      # This specifies the name of the file which will be created by the
      # analytic program.  This file will contain values that correspond
      # to the initial state of the atmosphere.  If this file already
      # exists, it will be replaced.

.run standard
    trop_ht  11.0
    dT/dz    -6.5
    thsfc   288.0
    uwnd      0.0
      # These lines specify that you want to create an atmosphere which
      # has a constant lapse rate (-6.5) up to the tropopause (11.0 km)
      # with a specified surface potential temperature (288 K) and a
      # horizontal wind speed (0).  The result is a stable atmosphere
      # that should remain constant with time.
```

Note: These commands can be in any order except for the .run command. The .run will start the processing of the information and so it must be the last command entered.

Note: After entering the .run command, a bunch of information will be displayed describing the data previously entered. Don't be alarmed. Continue to add the information regarding the type of analytic atmosphere to be created.

Note: There are other options as well, such as specifying the horizontal grid via the center points (.gridc), create terrain (.terrain hill) and creating lamb (.run lamb) or gravity (.run gravity) waves.

Modeldry

Step # 1 ... Compile the dry, simple model routines

```
% make modeldry
```

Step # 2 ... Start the modeldry program

```
% bin/modeldry
```

Step # 3 ... Type in the commands for the modeldry program

```
.time
   iyear   01   imonth  01    iday    01    ihour   00    imin    00
   fyear   01   fmonth  01    fday    01    fhour   03    fmin    00
      # These lines specify the starting date and the ending date.
      # The default starting date for the analytic program is 0000 UTC
      # on 01/01/01.

dt      0.0120
      # This specifies the time step to be used (120 seconds) during the
      # simulation.

.flow
    flow1  flow1.tmp
    flow2  flow2.tmp
      # This specifies the names of the temporary files used for storage
      # during the simulation.  The names are not as important as the
      # location.  These are very big files and should be placed in a
      # location where there is a lot of disk space available.

.start
    initial init.dat
      # These two lines specify the file that contains the initial state
      # of the atmosphere.  It should be the same file that was produced
      # by the analytic program.

.history model.his        hisfreq 30
      # This specifies the name of the file (model.his) that will
      # contain the data produced by the modeldry model during the
      # simulation cycle.  This file is known as the history file.  The
      # frequency of this output is also specified (30 time steps).

.hisout3 U  V  TV  P  HV
.hisout2 TER TERX TERY
      # These two lines specify the 3-D and 2-D variables that will be
      # saved to the history file (model.his).

.run  # This line tells the modeldry program to start making a forecast.
      # It is the last command to be used.
```

pltmout

Step # 1 ... Compile the routines that plot the history data

```
% make pltmout
```

Note: This program requires the NCAR graphics libraries. "pltmout" can't be run if this utility is not on your machine.

Step # 2 ... Start the pltmout program

```
% bin/pltmout
```

Step # 3 ... Type in the commands for the pltmout program

```
.help # This will give you a list of valid commands for the pltmout
      # program.

.data   model.his
      # This line tells the program the name of the file from which to
      # plot the data.

.time      0101010000
      # This line specifies the date of the data to be plotted.

.slice LAT      40.   50.
      # This line instructs the program to create cross-section maps at
      # 40 and 50 degrees north.  Any number of latitudes can be given.
      # If you have data that does not include 40N or 50N, then the
      # cross-sections plotted will refer to the closest latitude
      # available.

.map      THV     U
      # This line instructs the program to create maps of virtual
      # potential temperature (THV) and U-component horizontal winds (U).

.end  # This exits the program.
```

Note : The ".map", ".time" and ".lats" commands can be used repeatedly. However, the ".map" command will only produce maps corresponding to the last ".time" command given.

Note: To produce longitude cross-section maps, use ".lons" instead of ".lats".
To produce pressure level maps, use ".levels P" instead of ".lats".
To produce sigma level maps, use ".levels H" instead of ".lats".
To produce height level maps, use ".levels ZZ" instead of ".lats".
To produce surface maps, no level specifications are needed.
Use ".help" to see what other kinds of maps can be produced.

Plotting Graphs

The pltmout program doesn't create plots. All it does is create files which contain the plots in a coded form. These can only be read by an NCAR graphics utility. Thus, in order to view the plots which pltmout creates, one needs a NCAR graphics utility.

One possible NCAR graphics utility is called 'plt'. To enter the plot mode, type 'plt'. Another NCAR utility is called 'idt'. Once in the plot mode, the user can access metacode files (i.e., pltmout output), specify devices (i.e., screen or printer), specify frames to be plotted, etc.

Tutorial 5
A simpler procedure to run programs

This tutorial guides the user through a simpler procedure for running the LAMPS 90 package. For more complete information on this process, see the LAMPS 90 User's Guide.

This tutorial assumes that the user has already set-up the case directory as in tutorial 3 and ran the analytic, modeldry and pltmout programs as in tutorial 4.

After going through tutorial 4, the user has probably recognized how tedious it is to type in the commands for each program. This becomes even more tedious when one needs to re-run programs repeatedly, only changing small details in each run.
To make things easier for the user, one can create a file that contains all of the input for a particular program. This file is called the input file. Then, when the user runs a particular program (e.g., step # 2 in tutorial 4), all the user needs to do is let the program know that the input commands are to be read from the input file rather than from the screen. This is called re-directing input.

To re-direct input, use the less-than sign (<) followed by the name of the input file. For example, if a file named analytic.inp exists that contains all of the necessary input for the analytic program, the command

```
% bin/analytic < analytic.inp
```

will run the analytic program and take the input instructions from the analytic.inp program.

General Steps

For each LAMPS program, the general steps are...

1. Create the program (as in tutorial 4)

make <*name*>

2. Edit the appropriate command list

vi <*name*>.inp

3. Run the program

bin/<*name*> < <*name*>.inp

Important Notice

LAMPS 90 provides sample input files for each program. These sample input files are stored in the config/inp/ sub-directory of the main LAMPS 90 directory. It may be easier to make a copy of the appropriate sample input file and then edit it to fit your needs rather than starting from scratch.

Step - By - Step Guide

You should be located within your case directory.

Step # 1 ... Create an input file

```
% vi modeldry.inp
```

Use the vi editor to create the total list of commands as shown in tutorial 4.

Note: It is not necessary for the user to use the 'vi' editor. However, this editor is very common on UNIX machines. For information on the 'vi' editor, see doc/vi.help.

Note: One can first copy the sample input file from the config/inp/ sub-directory of the main LAMPS 90 directory and then edit this copy.

Step # 2 ... Delete the old history file (if any)

```
% rm model.his
```

Step # 3 ... Re-run the modeldry program using modeldry.inp as input

```
% bin/modeldry < modeldry.inp
```

Tutorial 6
A simpler way to create input files

This tutorial guides the user through a simpler way to create input files for all of the various programs of LAMPS 90. This tutorial assumes that the user has already set-up the case directory as in tutorial 3 and ran the analytic, modeldry and pltmout programs as in tutorial 4.

After going through tutorial 5, the user may have noticed that LAMPS 90 has many sample input files. This is because LAMPS 90 has many different programs and there needs to be one sample input file per program. Since it can be tedious to make copies of all the sample input files, the Setup input shell was created to copy all of the sample input files into a sub-directory of your case directory. The setup_input command also automatically makes the necessary changes to the input files according to parameters you specify.

The procedure outlined below mirrors tutorial 5 but uses the Setup shell to copy all of the sample input files to a sub-directory of your case directory.

Step - By - Step Guide

You should be located within your case directory.

<u>Step # 1</u> ... Set-up an inp/ sub-directory

```
% Setup input
```

Follow the directions. Specify the programs to be used appropriately and then just accept the default values.

<u>Step # 2</u> ... Edit an input file

```
% vi inp/modeldry.inp
```

Make sure everything looks okay (use <CTRL>-F and <CTRL>-B to page forward and backward). The end time will automatically be set to the start time so you must change the end time. Go to the fhour key word (e.g., use /fhour/<RETURN>wcw03<ESC>).

<u>Step # 3</u> ... Delete the old history file (if any)

```
% rm model.his
```

<u>Step # 4</u> ... Re-run the modeldry program using inp/modeldry.inp as input

```
% bin/modeldry < inp/modeldry.inp
```

Tutorial 7
The usual way to run programs

This tutorial guides the user through the usual steps used to run the LAMPS 90 package. For more complete information on this process, see the LAMPS 90 User's Guide.

This tutorial assumes that the user has already set-up the case directory as in tutorial 3, ran the analytic, modeldry and pltmout programs as in tutorial 4 and ran Setup and modeldry as in tutorial 6.

After going through tutorial 6, the user may be able to recognize the convenience of being able to do the following.

1. Take the information flashing across the screen during the running of each program and store it in a separate log file.
2. Automatically use the appropriate command list (i.e., input file).
3. Have the option to only run the program if the appropriate command list has been updated since the last run.
4. Have the option to only run the program if the program has been changed since the last run.
5. Take the gmeta file that is created, if any, and store in a separate grf sub-directory with a name corresponding to the program that created it (e.g., pltmout.gmeta).

General Steps

1. Make sure that the Makefile program knows where to find the input files.

   Within the Makefile file, there is a line that has "IDIR=inp". This states that the Makefile expects to find the input files in the sub-directory "inp/". If your input files are already in a sub-directory named "inp/", then nothing needs to be done. If not, either move all your input files into a sub-directory named "inp/" or make the appropriate change to the Makefile file. For example, if your input files are not in a sub-directory at all, change this line to read "IDIR=.". There are equivalent statements for the graphics sub-directory (GDIR) and log sub-directory (LDIR).

   Note: If you ran Setup to create the input files, the Makefile will automatically have IDIR set to the input directory designated during the Setup process.

2. Compile (if necessary) and run the program <name>.

   make <*name*> <*name*>.log

   This will compile the program if necessary and run it as before but will also:
   - take the information which usually flashes across the screen during the running of each program and store it in a file named "log/<name>.log".
   - automatically use the command list stored in the file named "<name>.inp".

- only run the program if one or more of the following conditions are met.
  - "<name>.inp" has been updated since the last run
  - "bin/<name>" has been changed since the last run
  - "log/<name>.log" is not present

Tutorial 3
Step - By - Step Guide

You should be located within your case directory.

<u>Step # 1</u> ... Compile the pltmout program

```
% make pltmout
```

Since the source hasn't changed since tutorial 4 (unless Setup case has been run again), no compilation should be necessary but it is good to do this step anyway just in case.

<u>Step # 2</u> ... Edit the appropriate input file

```
% vi inp/pltmout.inp
```

The default input file should be okay for now but check anyway.

<u>Step # 3</u> ... Run the pltmout program

```
% make pltmout.log
```

<u>Step # 4</u> ... Check the log file

```
% vi log/pltmout.log
```

Always check to make sure that there was a normal termination and that there were no WARNING or ERROR messages. Also check to make sure that everything went as expected.

<u>Step # 5</u> ... Check the graphics

```
% idt grf/pltmout.gmeta
```

Notes

Note: To force the program to run even if none of the conditions in statement 3 above are met, add the string "FORCE=force" (including quotes). For example,

```
      make <name>.log "FORCE=force"
```

Note: Multiple procedures can be done at once. For example,

```
      make analytic analytic.log modeldry.log "FORCE=force"
```

Note: If the program bombs during run-time, a temporary log file is created named <name>.log.tmp. Refer to this file for more information regarding why the program bombed.

Tutorial 8
Using .cfg files

This tutorial instructs the user about using .cfg files.

This tutorial assumes that the user has already set-up the case directory as in tutorial 3, ran the analytic, modeldry and pltmout programs as in tutorial 4, ran Setup and modeldry as in tutorial 6 and read tutorial 7.

Although there are many options that can be specified via the input file, there are other options that can only be specified via a file that contains C-preprocessor macro definitions. This file is then inserted into the source code during the compilation stage. The C-preprocessor macros are used for

1. Turning on and off flags (depending on the flag, different parts of the code will be compiled), and
2. Expanding variable names as functions.

In general, C-preprocessor macros are used rather than IF statements because IF statements can be costly time-wise.

Some of the options available through the use of cfg files is to change the boundary conditions (e.g., from sponge-type lateral boundaries to cyclic or time-dependent boundaries) and add additional write statements in order to debug a code. Tutorial 8 describes the steps needed to do the latter.

Step - By - Step Guide

You should be located within your case directory.

Step # 1 ... Edit the model .cfg file

```
% vi cfg/modelF.cfg
```

Define the trace macro. Go to the line that defines trace (e.g., /trace/<RETURN>) and change the #undef to #define (e.g., bcwdefine<ESC>).

Step # 2 ... Recompile the modeldry program

```
% make modeldry
```

Step # 3 ... Edit the input file

```
% vi inp/modeldry.inp
```

Defining trace causes a statement to be written to the log file every time the program enters a different subroutine. As a result, the log file becomes huge. Edit the modeldry input file and change the final time to be only a couple of time steps past the initial time (e.g., 04). Make sure the final hour is reset to 00.

Step # 4 ... Delete the old history file (if any)

```
% rm model.his
```

Step # 5 ... Re-run the modeldry program

```
% make modeldry.log
```

Step # 6 ... Check the log file

```
% vi log/modeldry.log
```

If all went well, then the log file should be rather large because every time the program entered a different subroutine, a statement to that effect is sent to the log file.

Step # 7 ... Change the model .cfg file back again

```
% vi cfg/modelF.cfg
```

Undefine the trace macro.

Step # 8 ... Change the input file back again

```
% vi inp/modeldry.inp
```

Tutorial 9
Making changes to the source code

This tutorial instructs the user about the steps needed to make changes to the source code. LAMPS 90 is really a research simulation model rather than a prediction model. As such, users usually want to use LAMPS 90 to test how a certain change affects the simulation.

This tutorial assumes that the user has already set-up the case directory as in tutorial 3, ran the analytic, modeldry and pltmout programs as in tutorial 4, ran Setup and modeldry as in tutorial 6 and read tutorials 7 and 8. This tutorial also assumes that you are comfortable with a text editor.

After going through tutorial 8, the user may want to make changes that aren't available through changing the input or .cfg files. The only option is for the user to use his/her own version of the source code.

*** * * * READ THIS !!! * * * ***

Since the source codes in the main LAMPS 90 directory are being used by each on the computer, these files SHOULD NOT BE TOUCHED. To make changes, each user MUST make a copy of the source code to be changed and put it in their own directory.

*** * * * READ THIS !!! * * * ***

Step - By - Step Guide

You should be located within your case directory.

Step # 1 ... Create a directory for your source code

```
% mkdir src
```

Step # 2 ... Change your working directory

```
% chdir src
```

Step # 3 ... Create a directory for the particular program to be changed

```
% mkdir modeldry
```

Step # 4 ... Change your working directory

```
% chdir modeldry
```

Step # 5 ... Create a new pblphys.pre file

```
% vi pblphys.pre
```

The pblphys.pre file contains the subroutine diffcoefs. For the dry, simple model, the diffcoefs subroutine is very short. All it does is set all of the diffusion coefficients (Km and Kth) to unity. A simple listing of the program follows.

```
#include "generic.cfg"
#include "case.cfg"
#include "merge.cfg"
#include "modelF.cfg"
#include "model_dry.cfg"

      subroutine diffcoefs ( islab, iline )

#include "flow_common.h"
#include "model_common.h"

      integer islab, iline

      do kv = 1, kmax
         zkm_new  ( i, j, kv ) = 1.
         zkth_new ( i, j, kv ) = 1.
      end do

      end
```

Your own version of pblphys.pre can be exactly as above or it can set the diffusion coefficients to some other value.

Note: The version of pblphys.pre in the main LAMPS 90 directory has much more code in it that this in order to have the option available to mimic the formulation of the full physics model (via the macro use_pbl).

<u>Step # 6</u> ... Change your working directory

```
% chdir ../..
```

<u>Step # 7</u> ... Edit the appropriate program makefile

```
% vi mks/modeldry.mk
```

Change the location of the pblphys.pre file. The pblphys.pre file is initially specified to be in the $(TRGT_DIR) directory. TRGT_DIR is a makefile macro that is set to the src/modeldry sub-directory of the main LAMPS 90 directory. You need to change this line to represent the src/modeldry sub-directory of your case directory. To do this, replace $(TRGT_DIR) with $(CASE_TRGT_DIR), which is also predefined for you.

<u>Step # 8</u> ... Delete the old history file (if any)

```
% rm model.his
```

<u>Step # 9</u> ... Compile and run the source code

```
% make modeldry modeldry.log
```

<u>Step # 10</u> .. Check the log file

```
% vi log/modeldry.log
```

<u>Step # 11</u> .. Run the pltmout program

```
% make pltmout.log "FORCE=force"
```

Note: The "FORCE=force" is necessary because despite not changing the input file nor the executable, we still want to run pltmout again.

Tutorial 10
Using the data volume utility

This tutorial shows the user how to go about checking the contents of a data volume. This tutorial assumes that the user has already read tutorials 1 through 9 and produced a history file named model.his.

To save space, LAMPS 90 uses a file structure that compresses data. This file structure is called the data volume. The history file created by the modeldry and model programs are in the data volume format. One of the records in the history file is called a header record. It contains information on what variables and such are present within the data volume. Since the data volume uses a packed data format, one cannot just use vi to view the contents of the header record. Instead, one must use a data volume utility called dvutil.

Step - By - Step Guide

You should be located within your case directory.

Step # 1 ... Compile the dvutil program

```
% make dvutil
```

Step # 2 ... Start the dvutil program

```
% bin/dvutil
```

Step # 3 ... Type in commands for the dvutil program

```
.help # This command isn't necessary but it will list out all of the
      # valid commands for the dvutil program.

.open   model.his

.listhdr

.list

.end
```

Notes

Note: To extract only certain times of the history file, use the .copy command. Be sure to copy the header file as well so that any program reading the new history file can know what is in each record.