NCAR-WRF on Raspberry Pi Cluster

System Environment Tests

First and foremost, it is very important to have a gfortran compiler, as well as gcc and cpp.

To test whether these exist on the system, type the following:

```
$ which gfortran
$ which cpp
$ which gcc
```

If you have these installed, you should be given a path for the location of each.

We recommend using gfortran version 4.4.0 or later. To determine the version of gfortran you have, type:

```
$ gcc --version
```

Create a new, clean directory called Build_WRF, and another one called TESTS.

```
$ cd /software

$ mkdir ncar-wrf_3.8.1

$ cd ncar-wrf_3.8.1

$ mkdir TESTS build
```

There are a few simple tests that can be run to verify that the fortran compiler is built properly, and that it is compatible with the C compiler. Below is a tar file that contains the tests. Download the tar file and place it in the TESTS directory.

Fortran and C Tests Tar File

```
$ cd TESTS

$ wget http://www2.mmm.ucar.edu/wrf/OnLineTutorial/compile_tutorial/tar_files/Fortran_C_tests.tar
```

To unpack the tar file, type:

```
$ tar xf Fortran_C_tests.tar
```

There are 7 tests available, so start at the top and run through them, one at a time.

Test #1: Fixed Format Fortran Test: TEST_1_fortran_only_fixed.f

Type the following in the command line:

```
$ gfortran TEST_1_fortran_only_fixed.f
```

Now type:

```
$ ./a.out
```

Output:

```
SUCCESS test 1 fortran only fixed format
```

Test #2: Free Format Fortran: TEST_2_fortran_only_free.f90

Type the following in the command line:

```
$ gfortran TEST_2_fortran_only_free.f90
```

and then type:

```
$ ./a.out
```

Output:

```
Assume Fortran 2003: has FLUSH, ALLOCATABLE, derived type, and ISO C Binding
SUCCESS test 2 fortran only free format
```

Test #3: C: TEST_3_c_only.c

Type the following in the command line:

```
$ gcc TEST_3_c_only.c
```

and then type:

```
$ ./a.out
```

The following should print out to the screen:

```
SUCCESS test 3 c only
```

Test #4: Fortran Calling a C Function:
TEST_4_fortran+c_c.c, and TEST_4_fortran+x_f.f90

Type the following in the command line:

```
$ gcc -c TEST_4_fortran+c_c.c
```

and then type:

```
$ gfortran -c TEST_4_fortran+c_f.f90
```

and then:

```
$ gfortran TEST_4_fortran+c_f.o TEST_4_fortran+c_c.o
```

and then issue:

```
$ ./a.out
```

Output:

```
C function called by Fortran
Values are xx = 2.00 and ii = 1
SUCCESS test 4 fortran calling c
```

In addition to the compilers required to manufacture the WRF executables, the WRF build system has scripts as the top level for the user interface. The WRF scripting system uses, and therefore having the following is necessary:

- csh
- perl
- sh

To test whether these scripting languages are working properly on the system, there are 3 tests to run. These tests were included in the "Fortran and C Tests Tar File".

Test #5: csh in the command line, type:

Error:

```
-bash: ./TEST_csh.csh: /bin/csh: bad interpreter: No such file or directory
```

Resolution

```
# apt install csh
```

Actual test:

```
$ ./TEST_csh.csh
```

Output:

```
SUCCESS csh test
```

Test #6: perl in the command line, type:

```
$ ./TEST_perl.pl
```

Output:

```
SUCCESS perl test
```

Test #7: sh in the command line, type:

```
$ ./TEST_sh.sh
```

Output:

```
SUCCESS sh test
```

Finally, inside the scripts are quite a few UNIX commands that are available regardless of which shell is used. The following standard UNIX commands are mandatory:

ar head sed
awk hostname sleep
cat ln sort
cd ls tar
cp make touch
cut mkdir tr
expr mv uname
file nm wc
grep printf which
gzip rm m4

Building Libraries

Before getting started, you need to make another directory.

Edit the *.bashrc* file to add environment variables:

```
# nano ~/.bashrc
```

Add the following to the end of the file:

```
# NETCDF-4.1.3
export DIR=/software/lib
export CC=gcc
export CXX=g++
export FC=gfortran
export F77=gfortran

export PATH=$DIR/netcdf_4.1.3/install/bin:$PATH
export NETCDF=$DIR/netcdf_4.1.3/install

# MPICH-3.04
export PATH=$DIR/mpich_3.0.4/install/bin:$PATH

# Zlib-1.2.7
export LDFLAGS=-L$DIR/grib2/lib
export CPPFLAGS=-I$DIR/grib2/include
```

Depending on the type of run you wish to make, there are various libraries that should be installed. Below are 5 libraries.

mpich-3.0.4

netcdf-4.1.3

Jasper-1.900.1

libpng-1.2.50

zlib-1.2.7

It is important to note that these libraries must all be installed with the same compilers as will be used to install WRFV3 and WPS.

NetCDF: This library is always necessary!

Set environment variables:

```
$ export DIR=/software/lib
$ export CC=gcc
$ export CXX=g++
$ export FC=gfortran
$ export F77=gfortran
```

Create installation directory:

```
$ cd $DIR
$ mkdir -p $DIR/netcdf_4.1.3/build
$ mkdir -p $DIR/netcdf_4.1.3/install
$ cd netcdf_4.1.3
```

Get library files:

```
$ wget http://www2.mmm.ucar.edu/wrf/OnLineTutorial/compile_tutorial/tar_files/netcdf-4.1.3.tar.gz
```

Untar files:

```
$ tar xzvf netcdf-4.1.3.tar.gz
$ cd build
```

Install:

```
$ $DIR/netcdf_4.1.3/netcdf-4.1.3/configure --prefix=$DIR/netcdf_4.1.3/install --disable-dap --disable-netcdf-4 --disable-shared

$ make
$ make install
$ make check
```

Activate environment variables:

```
$ export PATH=$DIR/netcdf_4.1.3/install/bin:$PATH
$ export NETCDF=$DIR/netcdf_4.1.3/install
```

Reset filesystem location for next library install:

```
$ cd $DIR
```

MPICH: This library is necessary if you are planning to build WRF in parallel. If your machine does not have more than 1 processor, or if you have no need to run WRF with multiple processors, you can skip installing MPICH.

In principle, any implementation of the MPI-2 standard should work with WRF; however, we have the most experience with MPICH, and therefore, that is what will be described here.

Assuming all the 'export' commands were already issued while setting up NetCDF, you can continue on to install MPICH, issuing each of the following commands:

NOTE: This is only required if no other MPICH or MPI library is installed

Create installation directory:

```
$ mkdir -p $DIR/mpich_3.0.4/build
$ mkdir -p $DIR/mpich_3.0.4/install
$ cd mpich_3.0.4
```

Get library files:

```
$ wget http://www2.mmm.ucar.edu/wrf/OnLineTutorial/compile_tutorial/tar_files/mpich-3.0.4.tar.gz
```

Untar files:

```
$ tar xzvf mpich-3.0.4.tar.gz
$ cd build
```

Install:

```
$ $DIR/mpich_3.0.4/mpich-3.0.4/configure --prefix=$DIR/mpich_3.0.4/install

$ make
$ make install
```

Set environment variables:

```
$ export PATH=$DIR/mpich_3.0.4/install/bin:$PATH
```

Reset filesystem location for next library install:

```
$ cd $DIR
```

zlib: This is a compression library necessary for compiling WPS (specifically ungrib) with GRIB2 capability

Assuming all the "export" commands from the NetCDF install are already set, you can move on to the commands to install zlib.

Set environment variables:

```
$ export LDFLAGS=-L$DIR/grib2/lib
$ export CPPFLAGS=-I$DIR/grib2/include
```

Create installation directory:

```
$ mkdir -p $DIR/grib2
$ cd grib2
```

Get library files:

```
$ wget http://www2.mmm.ucar.edu/wrf/OnLineTutorial/compile_tutorial/tar_files/zlib-1.2.7.tar.gz
```

Untar files:

```
$ tar xzvf zlib-1.2.7.tar.gz
```

Install:

```
$ cd zlib-1.2.7

$ ./configure --prefix=$DIR/grib2

$ make
$ make install
```

Reset filesystem location for next library install:

```
$ cd $DIR
```

libpng: This is a compression library necessary for compiling WPS (specifically ungrib) with GRIB2 capability

Assuming all the "export" commands from the NetCDF install are already set, you can move on to the commands to install zlib.

Create installation directory:

```
$ mkdir -p $DIR/libpng_1.2.50
$ cd libpng_1.2.50
```

Get library files:

```
$ wget http://www2.mmm.ucar.edu/wrf/OnLineTutorial/compile_tutorial/tar_files/libpng-1.2.50.tar.gz
```

Untar files:

```
$ tar xzvf libpng-1.2.50.tar.gz
```

Install:

```
$ cd libpng-1.2.50

$ ./configure --prefix=$DIR/grib2
```

```
$ make
$ make install
```

Reset filesystem location for next library install:

```
$ cd $DIR
```

JasPer: This is a compression library necessary for compiling WPS (specifically ungrib) with GRIB2 capability

Assuming all the "export" commands from the NetCDF install are already set, you can move on to the commands to install zlib.

Create installation directory:

```
$ mkdir -p $DIR/jasper_1.900.1
$ cd jasper_1.900.1
```

Get library files:

```
$ wget http://www2.mmm.ucar.edu/wrf/OnLineTutorial/compile_tutorial/tar_files/jasper-1.900.1.tar.gz
```

Untar files:

```
$ tar xzvf jasper-1.900.1.tar.gz
```

Install:

```
$ cd jasper-1.900.1

$ ./configure --prefix=$DIR/grib2

$ make
$ make install
```

Reset filesystem location for next library install:

```
$ cd $DIR
```

M4

Install m4 library:

```
# apt install m4
```

HDF5

Create installation directory:

```
$ mkdir -p $DIR/hdf5_1.10.1/install
$ cd hdf5_1.10.1
```

Get library files:

```
$ wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.10/hdf5-1.10.5/src/hdf5-1.10.5.tar.gz
```

Untar files:

```
$ tar xvfz hdf5-1.10.5.tar.gz
```

Install:

```
$ cd hdf5-1.10.5

$ $DIR/hdf5_1.10.5/hdf5-1.10.5/configure --prefix=$DIR/hdf5_1.10.5/install

$ make
```

Reboot before testing:

```
# reboot
```

Complete test and install: (Testing will take quite some time to complete: 3+ hours)

```
$ make check
$ make install
```

Library Compatibility Tests

Once the target machine is able to make small Fortran and C executables (what was verified in the System Environment Tests section), and after the NetCDF and MPI libraries are constructed (two of the libraries from the Building Libraries section), to emulate the WRF code's behavior, two additional small tests are required. We need to verify that the libraries are able to work with the compilers that are to be used for the WPS and WRF builds. Below is a tar file that contans these tests.

Download this tar file and place it in the TESTS directory:

Fortran_C_NETCDF_MPI_tests.tar

```
$ cd /software/ncar-wrf_3.8.1/TESTS

$ wget http://www2.mmm.ucar.edu/wrf/OnLineTutorial/compile_tutorial/tar_files/Fortran_C_NETCDF_MPI_tests.tar
```

To unpack the tar file, type:

```
$ tar xf Fortran_C_NETCDF_MPI_tests.tar
```

There are 2 tests:

Test #1: Fortran + C + NetCDF

The NetCDF-only test requires the include file from the NETCDF package be in this directory.

Copy the file here:

```
$ cp ${NETCDF}/include/netcdf.inc .
```

Compile the Fortran and C codes for the purpose of this test (the -c option says to not try to build an executable).

Type the following commands:

```
$ gfortran -c 01_fortran+c+netcdf_f.f
$ gcc -c 01_fortran+c+netcdf_c.c
$ gfortran 01_fortran+c+netcdf_f.o  01_fortran+c+netcdf_c.o -L${NETCDF}/lib -lnetcdff -lnetcdf

$ ./a.out
```

Output:

```
C function called by Fortran
Values are xx = 2.00 and ii = 1
SUCCESS test 1 fortran + c + netcdf
```

Test #2: Fortran + C + NetCDF + MPI

The NetCDF+MPI test requires include files from both of these packages be in this directory, but the MPI scripts automatically make the mpif.h file available without assistance, so no need to copy that one.

Copy the NetCDF include file here:

```
$ cp ${NETCDF}/include/netcdf.inc .
```

Note that the MPI executables mpif90 and mpicc are used below when compiling.

Issue the following commands:

```
$ mpif90 -c 02_fortran+c+netcdf+mpi_f.f
$ mpicc -c 02_fortran+c+netcdf+mpi_c.c
$ mpif90 02_fortran+c+netcdf+mpi_f.o 02_fortran+c+netcdf+mpi_c.o -L${NETCDF}/lib -lnetcdff -lnetcdf

$ mpirun ./a.out
```

Output:

```
C function called by Fortran
Values are xx = 2.00 and ii = 1
status = 2
SUCCESS test 2 fortran + c + netcdf + mpi
```

Building WRFV3

Before compiling WRF on a Raspberry Pi you will need to expand the swap file space to allow for low system memory.

Find the USB drive or external hard drive:

```
$ lsblk
```

It is most likely device *sda1*. Its physical location on the filesystem is */dev/sda1*.

Create a filesystem on the drive:

```
# mkfs.vfat /dev/sda1
```

Now edit the */etc/dphys-swapfile* file:

```
# nano /etc/dphys-swapfile
```

Change:

```
CONF_SWAPSIZE=1024
```

To:

```
CONF_SWAPSIZE=2048
```

Restart *dphys-swapfile*:

```
# /etc/init.d/dphys-swapfile stop
```

```
# /etc/init.d/dphys-swapfile start
```

After ensuring that all libraries are compatible w ith the compilers, you can now prepare to build WRFV3. If you do not already have a WRFV3 tar file, you can find it below.

Dow nload that file and unpack it in the *build* directory.

WRFV3.8.1

```
$ cd /software/ncar_wrf-3.8.1/build

$ wget http://www2.mmm.ucar.edu/wrf/src/WRFV3.8.1.TAR.gz

$ tar xfvz WRFV3.8.1.TAR.gz
```

Go into the WRFV3 directory:

```
$ cd WRFV3
```

Install required WRF-Chem package:

```
$ wget http://www2.mmm.ucar.edu/wrf/src/WRFV3-Chem-3.8.1.TAR.gz

$ tar xvfz WRFV3-Chem-3.8.1.TAR.gz
```

Setup WRF for Raspberry Pi. Edit the configuration files to conform to Raspberry Pi.

```
# nano arch/configure_new.defaults
```

Use `Ctrl+W` then `Ctrl+R` to find and replace the follow ing:

```
x86_64
```

With:

```
armv7l
```

The end character is a low ercase L.

Press ***A*** to replace all.

Save and exit

Configure Jasper environment variables:

```
$ export JASPERLIB=$DIR/grib2/lib
$ export JASPERINC=$DIR/grib2/include
```

Create a configuration file for your computer and compiler:

```
$ ./configure
```

You will see various options. Choose the option that lists the compiler you are using and the way you wish to build WRFV3 (i.e., serially or in parallel). Although there are 3 different types of parallel (smpar, dmpar, and dm+sm), we have the most experience with dmpar and typically recommend choosing this option.

For the Raspberry Pi you will pick ***34 dmpar gfortran***.

Accept the default on nesting.

Once your configuration is complete, you should have a configure.wrf file, and you are ready to compile. To compile WRFV3, you will need to decide which type of case you wish to compile.

The options are listed below:

- em_real (3d real case)
- em_quarter_ss (3d ideal case)
- em_b_wave (3d ideal case)
- em_les (3d ideal case)
- em_heldsuarez (3d ideal case)
- em_tropical_cyclone (3d ideal case)
- em_hill2d_x (2d ideal case)
- em_squall2d_x (2d ideal case)
- em_squall2d_y (2d ideal case)
- em_grav2d_x (2d ideal case)
- em_seabreeze2d_x (2d ideal case)
- em_scm_xy (1d ideal case)

Usage:

```
$ ./compile <case_name> >& compile.log
```

Where <case_name> is one of the options listed above.

Compilation should take about 30-45 minutes.

Once the compilation completes, to check whether it was successful, you need to look for executables in the WRFV3/main directory:

```
$ ls -ls main/*.exe
```

If you compiled a real case, you should see:

```
wrf.exe (model executable)
real.exe (real data initialization)
ndown.exe (one-way nesting)
tc.exe (for tc bogusing--serial only)
```

If you compiled an idealized case, you should see:

```
wrf.exe (model executable)
ideal.exe (ideal case initialization)
```

These executables are linked to 2 different directories:

WRFV3/run
WRFV3/test/em_real

You can choose to run WRF from either directory.

Building WPS

After the WRF model is built, the next step is building the WPS program (if you plan to run real cases, as opposed to idealized cases). The WRF model MUST be properly built prior to trying to build the WPS programs. Below is a tar file containing the WPS source code.

Download that file and unpack it in the Build_WRF directory:

WPSV3.8.1

```
$ cd /software/ncar-wrf_3.8.1/build

$ wget http://www2.mmm.ucar.edu/wrf/src/WPSV3.8.1.TAR.gz

$ tar xfvz WPSV3.8.1.TAR.gz
```

Go into the WPS directory:

```
$ cd WPS
```

Edit the configuration files to work with Raspberry Pi:

```
$ cd arch

# nano configure.defaults
```

Use `Ctrl+W` then `Ctrl+R` to find and replace the following:

```
i486 i586 i686
```

With:

```
armv7l
```

The end character is a lowercase L.

Press ***A*** to replace all.

Save and exit.

Similar to the WRF model, make sure the WPS directory is clean, by issuing:

```
$ cd ..

$ ./clean
```

The next step is to configure WPS, however, you first need to set some paths for the ungrib libraries:

```
$ export JASPERLIB=$DIR/grib2/lib
$ export JASPERINC=$DIR/grib2/include
```

and then you can configure:

```
$ ./configure
```

You should be given a list of various options for compiler types, whether to compile in serial or parallel, and whether to compile ungrib with GRIB2 capability. Unless you plan to create extremely large domains, it is recommended to compile WPS in serial mode, regardless of whether you compiled WRFV3 in parallel. It is also recommended that you choose a GRIB2 option (make sure you do not choose one that states "NO_GRIB2"). You may choose a non-grib2 option, but most data is now in grib2 format, so it is best to choose this option. You can still run grib1 data when you have built with grib2.

Choose the option that lists a compiler to match what you used to compile WRFV3, serial, and grib2.

The option used for the Raspberry Pi cluster is option 13 for serial gfortran.

Note: ***The option number will likely be different than the number you chose to compile WRFV3***

the metgrid.exe and geogrid.exe programs rely on the WRF model's I/O libraries. There is a line in the configure.wps file that directs the WPS build system to the location of the I/O libraries from the WRF model:

```
WRF_DIR = ../../WRFV3
```

Above is the default setting. As long as the name of the WRF model's top-level directory is "WRFV3" and the WPS and WRFV3 directories are at the same level (which they should be if you have followed exactly as instructed on this page so far), then the existing default setting is correct and there is no need to change it. If it is not correct, you must modify the configure file and then save the changes before compiling.

Before compiling you need to make one more change to the configuration file.

Edit *configuration.wps*:

```
# nano configure.wps
```

Under the *WRF_LIB* section:

Find:

```
-L$(NETCDF)/lib -lnetcdff -lnetcdf
```

Add *-lgomp* to the end:

```
-L$(NETCDF)/lib -lnetcdff -lnetcdf -lgomp
```

Now it will invoke OpenMP as needed.

We will also want to use MPI compiler.

Find:

```
DM_CC               = mpicc -cc=gcc
```

Change to:

```
DM_CC               = mpicc -cc=mpicc
```

You will also need to change the two lines for compression libraries.

Find the **second** set of the following:

```
COMPRESSION_LIBS
COMPRESSION_INC
```

Change to if it is not already set:

```
COMPRESSION_LIBS    = -L/software/lib/grib2/lib -ljasper -lpng -lz
COMPRESSION_INC     = -I/software/lib/grib2/include
```

Save and exit

You can now compile WPS:

```
$ ./compile >& compile.log
```

Compilation should only take a few minutes.

If the compilation is successful, there should be 3 main executables in the WPS top-level directory:

geogrid.exe
ungrib.exe
metgrid.exe

Verify that they are not zero-sized. To see file size, you can type:

```
$ ls -ls *.exe
```

Static Geography Data

The WRF modeling system is able to create idealized simulations, though most users are interested in the real-data cases. To initiate a real-data case, the domain's physical location on the globe and the static information for that location must be created. This requires a data set that includes such fields as topography and land use catergories. These data are available from the WRF dow nload page, but a tar file of the basic complete data set is also available below :

Geography Static Data

Dow nload the file, and place it in the Build_WRF directory.

Uncompress and un-tar the file:

```
$ cd /software/ncar-wrf_3.8.1/build

$ wget http://www2.mmm.ucar.edu/wrf/src/wps_files/geog_complete.tar.gz

$ tar xvfz geog_complete.tar.gz
```

When you untar the file, it w ill be called "geog."

Rename the file to "WPS_GEOG."

```
$ mv geog WPS_GEOG
```

The directory infomation is given to the geogrid program in the namelist.w ps file in the geogrid section:

```
$ geog_data_path = ´/software/ncar-wrf_3.8.1/build/WPS_GEOG´
```

The data expands to approximately 10 GB. This data allow s a user to run the geogrid.exe program.

Real-time Data

For real-data cases, the WRF model requires up-to-date meteorological information for both an initial condition and also for lateral boundary conditions. This meteorological data is traditionally a Grib file that is provided by a previously run external model or analysis. For a semi-operational set-up, the meteorological data is usually sourced from a global model, which permits locating the WRF model's domains anywhere on the globe.

The National Centers for Environmental Prediction (NCEP) run the Global Forecast System (GFS) model four times daily (initializations valid for 0000, 0600, 1200, and 1800 UTC). This is a global, isobaric, 0.5 degree latitude/longitude, forecast data set that is freely available, and is usually accessible +4h after the initialization time period.

A single data file needs to be acquired for each requested time period. For example, if we would like hours 0, 6, and 12 of a forecast that is initialized 2014 Jan 31 at 0000 UTC, we need the following times:

- 2014013100 – 0 h
- 2014013106 – 6 h
- 2014013112 – 12 h

These translate to the following file names to access:

- gfs.2014013100/gfs.t00z.pgrb2.0p50.f000
- gfs.2014013100/gfs.t00z.pgrb2.0p50.f006
- gfs.2014013100/gfs.t00z.pgrb2.0p50.f012

Note that the initialization data and time (gfs.2014013100) remains the same, and that the forecast cycle remains the same (t00z). What is incremented is the forecast hour (f00, f06, f12).

Before obtaining the data, creat a directory in Build_WRF, called "DATA", and then go into that directory:

```
$ cd /software/ncar-wrf_3.8.1/build

$ mkdir DATA
$ cd DATA
```

A simple set of interactive commands to grab these files from the NCEP servers in real-time would look like (**Note: This is just an example time/date. Typically on the NCEP data servers, only the most recent 2-3 days are available at any given time. To use up-to-date real-time data, you will need to adjust the commands to reflect current date and time information**):

```
curl -s --disable-epsv --connect-timeout 30 -m 60 -u anonymous:USER_ID@INSTITUTION -o GFS_00 \
ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gfs.2014013100/gfs.t00z.pgrb2.0p50.f000

curl -s --disable-epsv --connect-timeout 30 -m 60 -u anonymous:USER_ID@INSTITUTION -o GFS_06h \
ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gfs.2014013100/gfs.t00z.pgrb2.0p50.f006

curl -s --disable-epsv --connect-timeout 30 -m 60 -u anonymous:USER_ID@INSTITUTION -o GFS_12h \
ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gfs.2014013100/gfs.t00z.pgrb2.0p50.f012
```

Typically these commands return a complete file within a few seconds. The files returned from these commands (GFS_00h, GFS_06h, GFS_12h) are Grib Edition 2 files, able to be directly used by the ungrib program.

You need to fill in the anonymous login information (which is not private, so there are no security concerns about leaving these scripts around). You will probably end up writing a short script to automatically increment the initialization time.

Run WPS and WRFV3

Below are basic instructions for running WPS and WRFV3. For more detailed information, please see the WRF-ARW Online Tutorial

> **Running WPS**

You are now ready to begin running WPS and WRFV3. Start by going to the WPS directory:

```
$ cd /software/ncar-wrf_3.8.1/build/WPS
```

Make any changes to the namelist.wps file, to reflect information for your particular run

Before running geogrid, make sure that you have your geog_data_path set to the location where you put your geography static data. Once that is set, you can run geogrid.

```
$ ./geogrid.exe >& geogrid.log
```

If you successfully created a geo_em* file for each domain, then you are ready to prepare to run ungrib. Start by linking in the input GFS data:

```
$ ./link_grib.csh <path_where_you_placed_GFS_files>
```

Then link to the correct Vtable (GFS, for this case):

```
$ ln -sf ungrib/Variable_Tables/Vtable.GFS Vtable
```

Then run the ungrib executable:

```
$ ./ungrib.exe
```

You should now have files with the prefix "FILE" (or if you named them something else, they should have that prefix)

You are now ready to run metgrid:

```
$ ./metgrid.exe >& metgrid.log
```

You should now have files with the prefix met_em* for each of the time periods for which you are running.

> **Running WRFV3**

You are now ready to run WRFV3. Move into the WRFV3 directory, and then into either the run/ directory, or the test/em_real/ directory:

```
$ cd ../WRFV3/run
```

or

```
$ cd ../WRFV3/test/em_real
```

Before running the "real" program, you need to make all necessary changes to reflect your particular case to the namelist.input file. Once that is complete, you need to copy or link your met_em* files into the working directory:

From the *test/em_real* directory:

```
$ ln -sf ../../../WPS/met_em* .
```

or

From the *run/* directory:

```
$ ln -sf ../../WPS/met_em* .
```

or, if you would rather copy the files in, instead of linking them, you can use the cp command, instead of the ln -sf command.

You can now run the "real" program. The command for running this may vary depending on your system and the number of processors you have available, but it should be something similar to:

```
mpirun -np 1 ./real.exe
```

Check the end of your "rsl" files to make sure the run was successful:
tail rsl.error.0000

If you see a "SUCCESS" in there, and you see a wrfbdy_d01 file, and wrfinput_d0* files for each of your domains, then the run was successful.

To run WRFV3, type something similar to:

```
$ mpirun -np 8 ./wrf.exe
```

Again, check your "rsl" file for "SUCCESS", and make sure you have all the wrfout* files you anticipated having. If so, the run was successful, and you are ready to do analysis for your project.

Where to get started

WRF online tutorial:
http://www2.mmm.ucar.edu/wrf/OnLineTutorial/index.htm

Katrina example:
http://www2.mmm.ucar.edu/wrf/OnLineTutorial/CASES/SingleDomain/index.html

Troubleshooting Section:

> ### The Node Randomly Reboots:
>
> This error is indicated by the node rebooting.Under intensive computation, the board needs a minimum amount of voltage to carry out these tasks. The node then gets stuck in a rebooting process, in which the node's image must be refreshed with a previously working image. This problem is a result of using a low voltage, micro USB cable to power on a pi board. Switch out USB cables, and try from a working SD card image.