WRF Simulation Notes

Nigel Woodhouse

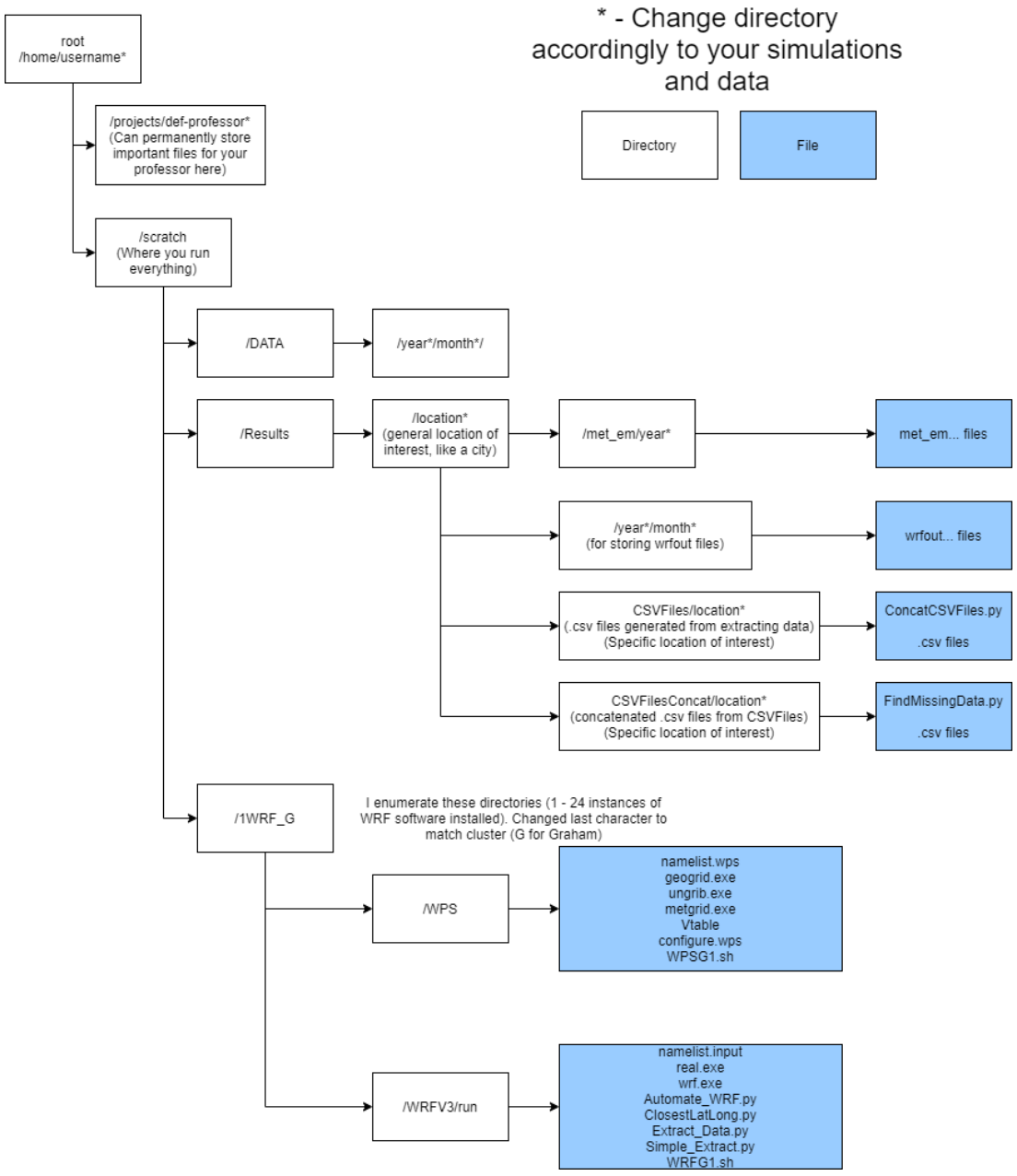
Updated January 31, 2021

Contents

All useful online resources are in the "Links and Additional Resources" section at the end of this document

Directory Layout

This flowchart is how I organized my directories and files and is followed throughout the documentation. However, you may make your modifications to suit your needs. Only notable files/directories are displayed.



Program Order

This should be roughly the order upon which programs are executed. They are either supplied by the professor, or on my GitHub: https://github.com/NigelWoodhouse

1. WPSG1.sh
2. WRFG1.sh -> Do a 10-minute simulation, use simple version
3. ClosestLatLong.py
4. Simple_Extract_Data.py
5. Automate_WRF.py
6. WRFG1.sh -> use automated version
7. Extract_Data.py
8. ConcatCSVFiles.py
9. FindMissingData.py

Text Editor and Connecting to Clusters

I recommend using **VS Studio Code** for connecting with the supercomputer clusters. It provides a basic file navigation, terminal commands, and the ability to program in a normal code editor rather than using nano or other Linux text editors. I am sure you can figure out how to install VS Code.

To SSH into the clusters, install the Vs Studio Code plugin **Remote-SSH**. Search the Extensions Marketplace in VS Code and install Remote-SSH. You will need to restart VS Code afterwards to implement it.

Open Vs Code again. Press **Ctrl-Shift-P** and select **Remote-SSH: Open configuration file** and open the **\.ssh\config** file. Add this to the file, creating a new segment for every host you wish to connect to.

```
Host Graham
    HostName graham.computecanada.ca
    User your_computecanada_user_name

Host Cedar
    HostName cedar.computecanada.ca
    User your_computecanada_user_name

Host Beluga
    HostName beluga.computecanada.ca
    User your_computecanada_user_name
```

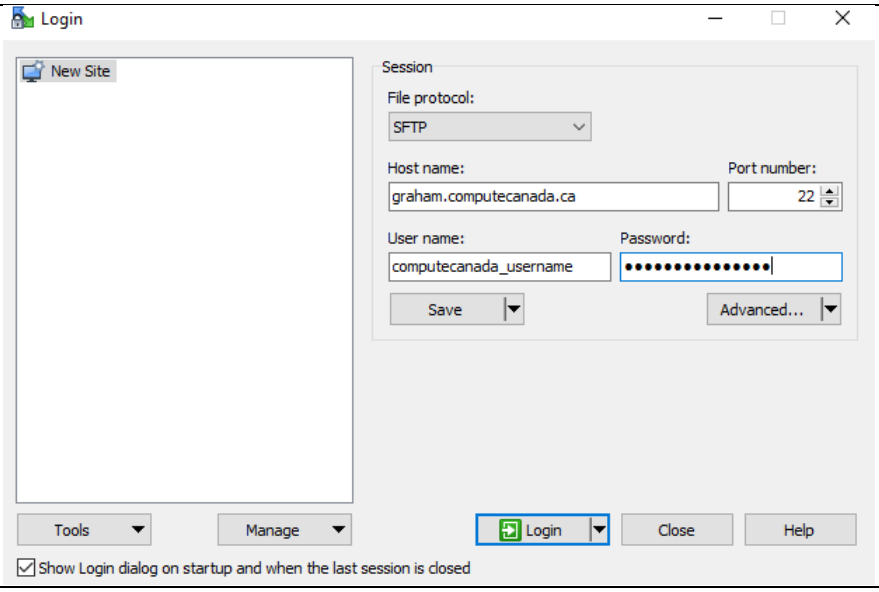
You need your professor/advisor to create an account for you on Compute Canada to access the clusters.

Hostnames we have access to on Compute Canada are Graham, Cedar, and Beluga. Hostname follows the same format, that is, the cluster address. User is your login to Compute Canada. To connect, there should be a green icon in the bottom left of the window. Click that icon, and that will prompt your Compute Canada password.

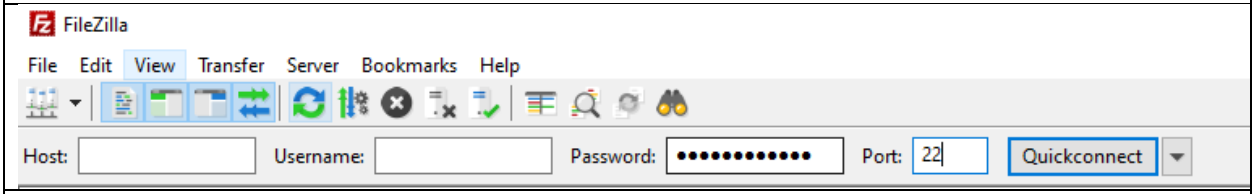
From personal experience, I have found Graham to be the most stable. It is the most widely used and therefore can experience long wait times when submitting jobs. Cedar and Beluga seem to have more tech issues (issues typing in the terminal, slow terminal responses, issues connecting), yet queue wait times appear to be shorter; this is entirely anecdotal; you should make your own judgements. Otherwise, all clusters behave roughly the same.

File Transfer Protocol (FTP) with clusters

You may wish to transfer files between the cluster and your personal computer, which can be done with FTP programs. I recommend using WinSCP and/or FileZilla. A quick Google search can land you on the downloads page, to where you download on your personal computer. To access the graham cluster and transfer files, for example,



WinSCP



FileZilla. Host -> sftp://graham.computecanada.ca. Username is your computecanada_username

Use Port 22 for all clusters.

I find the interface of WinSCP to be cleaner, but FileZilla is faster at transferring many or large files.

Installing WPS/WRF

I'm effectively going to condense the information from this link. Use it for more information on installing and recompiling. Additionally, I assume that you have some basic knowledge of Linux.

https://wiki.math.uwaterloo.ca/fluidswiki/index.php?title=WRF_Tutorial

Login to a cluster. The **scratch directory** is where you do everything on the cluster. I will create a directory named **1WRF_G**. Regardless, call it whatever you want. To make a new directory, use

mkdir directory_name

You can move into / change directory with

cd directory_name

I'll been using and installing WRF version 3.9.1.1 and WPS module 3.9.1.

To load modules, it follows the format:

```
module load module_name/version
```

To install WRF and WPS,

```
module load wrf/3.9.1.1
module load wps/3.9.1
```

You can now install the software packages into the directory.

```
cp -r $EBROOTWRF/WRFV3 .

cp -r $EBROOTWPS .
```

This creates a directory named *3.9.1* when installing WPS. You can move all the contents up a level and remove the directory.

(remove files using *rm filename*, remove directories using *rm -r filename*)

(move files using *mv filename destination*)

(I'll also mention here that * is a wildcard, that any string of characters can satisfy that spot; useful for moving many similarly named files)

Currently, the **1WRF_G** should contain these 4 directories:

```
easybuild, geog, WPS, WRFV3
```

Congrats, they are now installed.

Note: as you may have noticed, I am naming the main directories #WRF_G. Simulations are set up using namelist.wps and namelist.input files (we will talk about these later). I was never able to figure out how to modify these files in job arrays actively -> that is running multiple sims from the same software at the same time. I have 24 of the WPS and WRF installed on each cluster (though it may not be necessary to install so many WPS (maybe 4 max), as the sims run quickly and can utilize the same

geog directory). Additionally, I note which cluster I am on, which helps when you are bounding between different SSH instances.

Getting weather Data

Create a new directory in **scratch** to store the raw weather data. Let's name the directory **DATA**. I fuller created directories for each year, and further for each month, as an easy way to organize the data, and is beneficial for later. For example

scratch > DATA > 2020 > 202001 > store respective data here.

I have attempted to use two different weather repositories. There is NARR (North American Regional Reanalysis) and NAM (North American Mesoscale Forecast). NARR outputs data every 3 hours with a spatial resolution of 32km, and I believe outputs more data/variables (an excessive amount for the work I was interested in, but whatever). NAM outputs every 6 hours with a spatial resolution of 12km (not entirely sure of how many variables it can output, but I believe significantly fewer). So, take your pick; you may want to do more research for which would be better; either should be sufficient for most tasks.

NARR

For downloading NARR data, I found it best to use UCAR (you need to sign up for an account).

https://rda.ucar.edu/datasets/ds608.0/index.html#cgi-bin/datasets/getWebList?dsnum=608.0&action=customize&disp=

If that link doesn't work,

https://rda.ucar.edu/datasets/ds608.0/#!description

All the variables are wrapped up into one, so don't worry about selecting the ones you want; select all. Select the files you want to download. And then press **Create** on "Create a Unix script to read them from the Internet using Wget." That will open a bunch of text. Copy it.

Back in the location where you want to download the data, create a new file with *nano file_name*. To make a new executable, one method is:

```
nano get.exe
```

Paste the text. Replace the xxxxxx on line 19 with the password to your account on UCAR. Additionally, at the ***end*** of the file, type

```
rm get.exe
```

to remove the new executable file after running, for convenience sake. This file needs to be removed from the directory anyway, might as well do it now. Save the file.

Make the file executable by changing permissions:

```
chmod +x get.exe
```

Run it using

```
./get.exe
```

Now you have a bunch of .tar files. You do not have to untar these files; wps can handle them as is. However, with NAM, it contains forecast data in the download which you may wat to get rid of. To untar all the files in the directory, use

```
for a in `ls -1 *.tar`; do tar -xvf $a; done
```

and remove the .tar files with

```
rm *.tar
```

Congrats, you have downloaded NARR data.

NAM

Disclaimer: the methods I will describe below can be used to quickly download NAM data. However, I have noticed that they have some gaps in their data (missing days). For more complete data, I would recommend the method above for NARR.

You can download data from NOAA:

https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/north-american-mesoscale-forecast-system-nam

Clicking on the AIRS, you can select the times (highlight all 4 -> 00, 06, 12, 18) and the date range; this will create a webpage where you can download the data after the request is fulfilled. Upon completion write in the terminal in the directory you want to install in:

```
wget -e robots=off -r -l 1 -np 'http://example.com/folder/'
```

The link you want is the request generate by NOAA for data requested.

It will place the *.tar files in many subdirectories. You can move them up and delete the many new directories. Make sure that the **index** files are also removed.

Congrats, you have downloaded NAM data. (I recommend untarring the files and removing the forecast results files *001.grb, *002.grb, *003.grb, *006.grb. How to untar is in the section above describing how to get NARR data).

Alternatively, and faster (but may be missing data)

You can use the following command to download data from NOAA, as it is already indexed and stored. Just change the following to the date required (2 dates to change) -> downloads one month at a time. Run this in the folder you want the data stored in.

```
wget -e robots=off -r -np -l 2 -A _000.grb https://www.ncei.noaa.gov/data/north-american-mesoscale-model/access/historical/analysis/201401/; mv www.ncei.noaa.gov/data/north-american-mesoscale-model/access/historical/analysis/201401/**/* .; rm -r www.ncei.noaa.gov/
```

WPS Setup and Running

Linking Data

We will be operating in the WPS directory, for example:

scratch > 1WRF_G > WPS

First thing is to link the data you downloaded. Use:

```
./link_grib.chs ~/scratch/DATA/location_of_your_data/*
```

Next link the proper Vtable (Variable Table: it is how the software knows what variables mean what and how to organize it).

For NARR data, do:

```
ln -sf ungrib/Variable_Tables/Vtable.NARR Vtable
```

For NAM data, do:

```
ln -sf ungrib/Variable_Tables/Vtable.NAM Vtable
```

As you can see, it is easy to link the proper Vtable if a different data set is used.

namelist.wps

Ok, this gets a bit cumbersome, so I will do my best to explain. The namelist.wps file controls parameters for setting up the simulation, and the results will eventually be passed on to WRF. I will only highlight the lines of relevance.

```
 max_dom = 2,
```

This is the number of domains to be used in the simulation. Here I am only using 2 domains, as that is what worked well for my application. You have the option to use a 3rd domain or more, given the context of your problem.

```
 start_date = '2014-01-01_00:00:00','2014-01-01_00:00:00'

 end_date   = '2014-02-01_00:00:00','2014-02-01_00:00:00'
```

Pretty self explanatory; the date range of the simulation (and data linked).

```
 interval_seconds = 21600
```

This is the frequency upon which the raw weather data is logged, in seconds. NAM is 6 hours (21600 seconds), NARR is 3 hours (10800 seconds).

```
 parent_id       =  1,   1
```

How the software identifies its parent (outer domain).

```
 parent_grid_ratio =  1,  3
```

How spatially refined the domains are compared to their parent. Example, second domain is 3x more refined than its parent.

```
 i_parent_start   =  1,  42
```

```
j_parent_start   =  1,   42
e_we          = 125,   124
e_sn          = 125,   124
```

This is determining the size and positioning of the domains.

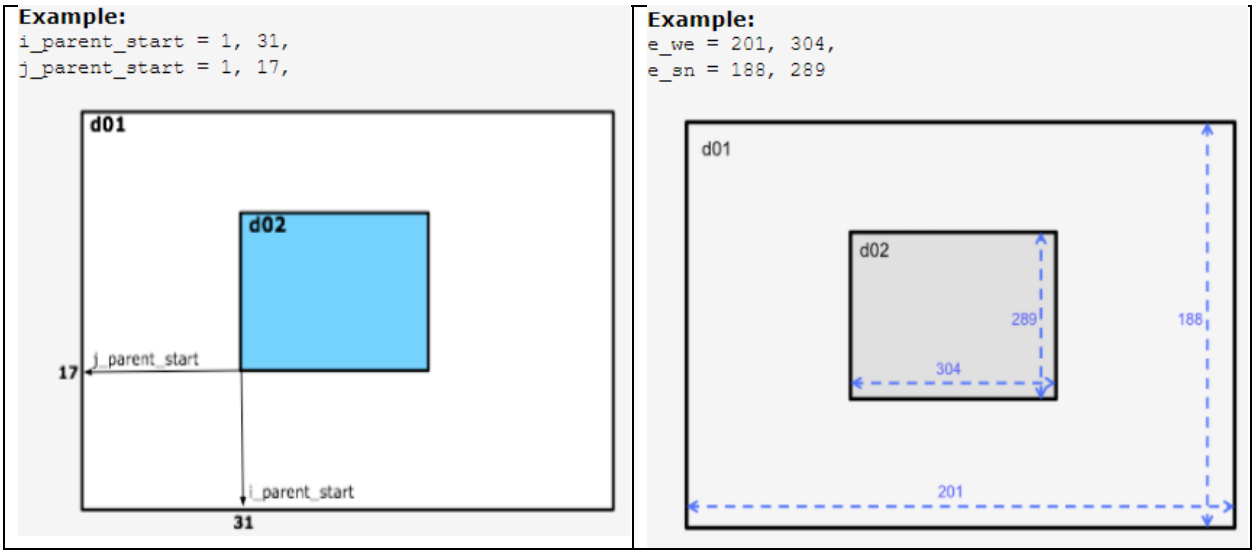
Let's starts with e_we (west_east) and e_sn (south_north). These represent the area that the domain will cover (size of domain), represented by the number of grid spaces. These values are dependent on the dx/dy values (explained later) and gird spacing ratios. It is **highly** recommended that these values are **greater than 100x100 for each domain**. The smallest domain **should** cover all (or at least most) of the area of interest.

Although, the larger your domains, the longer the time to simulate. With the parameters presented above, it takes ~10 hours to simulate 2 days, which is a lot. So I would suggest decreasing the values if time is of a constraint.

Additionally, "your nest domain dimensions will be (ending index - starting index)*3+1". In other words, "must be one greater than an integer multiple of the parent_grid_ratio of #."

The i_parent_start (east_west) and j_parent_start (north_south) describe the relative position with respect to its parent domain. The numbers refer to the number of grid spacing of the parent domain before the child domain starts.

The following two pictures illustrate well.



Best practices for domain

https://www2.mmm.ucar.edu/wrf/users/namelist_best_prac_wps.html

```
  dx = 4000,
  dy = 4000,
```

The spatial resolution of the domain, in meters, is derived from the dataset. For NAM, it is 12km (12,000 meters). For NARR it is 32km (32,000 meters). However, in both contexts, I do an initial refinement of 3X, so the dx and dy values I would give NARR is 10,800 meters, and NAM is 4,000 meters.

```
  ref_lat   =  53.518,
  ref_lon   = -113.488,
  truelat1  =  50.0,
  truelat2  =  56.0,
  stand_lon = -113.488,
```

ref_lat, ref_lon is the center of the location you are interested in. It is good practice that stand_lon is the same as ref_lon.

I'm not entirely sure what truelat1 and truelat2 do; I think it is how the top and bottom of the domain curve -> cannot place a flat surface on a spherical body. Make them roughly the latitude lines that the largest domain (Domain 01) curves along.

Additional note (not mandatory)

```
  geog_data_path = '../geog/'
```

If you are installing the software package multiple times (as I did), they can all use the same **geog** directory (it's a big directory). You can avoid installing it multiple times by installing segments of WPS, i.e.

```
cp -r $EBROOTWPS/WPS .
cp -r $EBROOTWPS/easybuild .
```

And change where the geog_data_path links to in the namelist.wps files.

Running WPS

I recommend running a quick 10 minute simulation in order to check domain and spacing before ramping things up and simulating all of your data.

Everything in WPS should be setup now.

Run these in order:

```
./geogrid.exe
./ungrib.exe
./metgrid.exe
```

If you have followed everything, you should be ok. At the end of each executable run, there is a successful message. If there is an error, it should tell you what is wrong or what you missed. Typically, an error arises cause you forgot to link data, link the Vtable, have the right date, or domain cannot be adequately constructed.

The output that you want is many met_em.d0...[date].nc files in the current directory. I would recommend moving all the met_em* files to a separate directory, from where you can soft link them later. For example, I have saved mine to the directory **~/scratch/Results/NAM/2014/**

```
mv met_em* ~/scratch/Results/NAM/2014
```

-> obviously, put in the correct date(s). We will soft link these files later for the WRF package to use.

NOTE

Sometime when running ungrib.exe with NAM data, the message "Subsoil level 300 in the GRIB2 file, was not found in the Vtable". Kind of know why, don't know how to fix. Not sure if it impacts results that I am interested in.

WRF Setup and Running

Navigate to the **WRFV3/run** directory.

Running WRF is straight forward at this point. The first thing we will do is to soft link the met_em* files that we produced from WPS. This can be done by running:

```
ln -s ~/scratch/Results/NARR/2014/* .
```

or wherever your met_em* files are stored.

A soft link effectively works as a pointer; the files do not reside in that directory but link to where they are stored, so you do not have to create multiple copies of the files.

The next step is to set up the file that establishes the simulation's parameters, **namelist.input**. I will try and do my best to describe the important points in the file. I will only highlight the ones I have changed from default parameters or have some importance.

```
&time_control
 run_days                 = 2,
 run_hours                = 00,
 run_minutes              = 0,
 run_seconds              = 0,
 start_year               = 2016, 2016,
 start_month              = 01, 01,
 start_day                = 01, 01,
 start_hour               = 00,   00,
 start_minute             = 00,   00,
 start_second             = 00,   00,
 end_year                 = 2016, 2016,
 end_month                = 01, 01,
 end_day                  = 03, 03,
 end_hour                 = 00,   00,
```

```
 end_minute                          = 00,   00,
 end_second                          = 00,   00,
```

This controls when the simulation begins, how long it runs for, and when it ends. What each parameter means should be self-evident.

I believe a simulation can have two ending clauses, either the simulation duration or the end date; I believe the software ends on whichever parameters comes first -> limiting condition.

It has been suggested to me to run simulations in a 2-day burst. The first day is treated as a warm-up to get the software to proper operating conditions, and then only take the second day. More extended duration simulations may generate more errors the further it goes in time?? -> I have not confirmed that to be true—just something to be aware of.

```
 interval_seconds                    = 21600
```

How frequently the data is outputted from the dataset; same as from WPS.

```
 history_interval                    = 5,   5,
```

How frequently data is outputted to the output file. As I want data in 5-minute increments, I have set both to 5.

```
 frames_per_outfile                  = 1000, 1000,
```

Control the output files' size -> how many time intervals/data points will be saved to the output file. If it exceeds the threshold, then it creates a new file. For only 2-day simulations, leaving at 1000 is fine.

```
 &domains
 time_step                           = 24,
 time_step_fract_num                 = 0,
 time_step_fract_den                 = 10,
```

Time step of simulation. It is recommended that the timestep be 6X of the dx value (in seconds from kilometers -> dx = 4000 meters = 4 km x 6 = 24 seconds). Make sure you have **some value other than 0** for **time_step_fract_den**, or else you end up with segmentation faults -> dividing by 0.

```
 max_dom                             = 2,
 e_we                                = 125,    124,
 e_sn                                = 125,    124,
```

These should be the same from WPS

```
 e_vert                              = 30,    30,
```

Number of vertical layers -> for both NAM and NARR, it is 30.

```
 p_top_requested                     = 10000,
```

I think this is the default parameter, and I believe the same for both NAM and NARR.

```
 num_metgrid_levels                  = 40,
```

```
 num_metgrid_soil_levels             = 4,
```

For NAM, num_metgrid_levels is 40, for NARR, num_metgrid_levels is 30.

```
 dx                                  = 4000, 1333.333,
 dy                                  = 4000, 1333.333,
 grid_id                             = 1,     2,
 parent_id                           = 0,     1,
 i_parent_start                      = 1,     42,
 j_parent_start                      = 1,     42,
 parent_grid_ratio                   = 1,     3,
 parent_time_step_ratio              = 1,     3,
```

These should be the same from WPS. You need to do the math for dx and dy -> take the initial dx and dy from WPS and divide by grid ratio.

```
 &physics
 physics_suite                       = 'CONUS'
 cu_physics                          = 0,     0,
 radt                                = 5,     5,
 bldt                                = 0,     0,
 cudt                                = 5,     5,
 icloud                              = 1,
 num_soil_layers                     = 4,
 num_land_cat                        = 21,
 sf_urban_physics                    = 0,     0,
```

'CONUS' is a default physics parameter setup, which should give good results. Two changes to this are the **cu_physics**; It is recommended that I add that line and change it to 0's when working with domains under 10,000-meter **dx/dy** values.

Another change is setting **radt** values to 5; this will call the radiation physics modules every 5 minutes (calls short/longwave radiation up/down) and write to the output file every 5 minutes, like all other parameters. I.E. make this value the same as **history_interval.**

The rest of the namelist.input file reamins as default parameters.

Running WRF

To run WRF, run

```
./real
```

If this runs fine with no error, then the rest should go well. I will describe checking errors in the following section. This executable should only take a few minutes to run.

Then run

```
./wrf.exe
```

This is the big one and will take a long time to run, as this is where most of the number-crunching happens. So sit back and relax. You've earned it.

After the simulation runs, it will leave you with wrfout_[date] files. (They should have a .nc [NetCDF file] extension, but don't for some reason). After which, we can extract the data from these files and store it in a .csv file.

WRF Errors

The outputs of both the **real.exe** and **wrf.exe** are saved to a file called **rsl.error.0000**. If there are any error messages, you will find them at the end of the file, and it tends to give you a decent description of what went wrong. Usually, it is because you did not link the met_em* files to the directory (or the ones with the proper date) or parameters in **namelist.input** and **namelist.wps** are different.

Beyond that, try to interpret what the error message says, use Google, or this forum.

https://forum.mmm.ucar.edu/phpBB3/viewforum.php?f=3&sid=35a12661f754ae59f8b21ed0ac723cf9

Dealing with Leap Years

Leap years are fun; it will only affect February for years divisible by 4.

By default, the WPS and WRF software is downloaded (by default) using a non-leap year calendar – I don't know why. The NAM and NARR datasets both contain leap-years. If this is not fixed, the software will notice data for February 29 exists and close.

To fix this, we need to recompile both pieces of software. For extra details about recompiling WPS and WRF, refer to this:

https://wiki.math.uwaterloo.ca/fluidswiki/index.php?title=WRF_Tutorial

Recompile WPS

Navigate to your **WPS** directory. Enter this into the terminal.

```
./clean -a
```

Wait for it to clean the files. You won't lose files you've put in there.

```
./configure
```

Choose option **19**. Wait for it to configure.

Edit the configure.wps file.

```
nano configure.wps
```

About 2/3 of the way down, look for **CPPFLAGS**. Remove **-DNO_LEAP_CALENDAR.** Save file. Compile,

```
./compile
```

Compiling takes a while, but afterwards, you are good to go.

Recompile WRF

Navigate to your **WRFV3** directory (Not WRFV3/run). Enter this into the terminal.

```
./clean -a
```

Wait for it to clean the files. You won't lose files you've put in the WRFV3 or WRFV3/run directories.

```
./configure
```

Choose option **15, then 1** (1=basic). Wait for it to configure.

Edit the configure.wrf file.

```
nano configure.wrf
```

About 1/3 of the way down, look for **ARCH_LOCAL**. Remove **-DNO_LEAP_CALENDAR.** Save file. Compile,

```
./compile em_real
```

Compiling takes a long time, but afterwards, you are good to go.

Using Python – my language of choice

I have found that the easiest way to extract NetCDF files is using python, though there are online resources. I will describe codes here in a bit. You need to set up Python 3 on your system. The best way to do it is through this documentation here:

https://docs.computecanada.ca/wiki/Python

Use pip to install python libraries. I recommend installing NumPy, SciPy, pandas, NetCDF4 (this one has been annoying to install sometimes for an unknown reason).

If you are running bash programs on the cluster, you will need

```
#!/bin/bash
```

At the top of your .sh files.

If you are running Python programs on the cluster, you will need

```
#!/usr/bin/env python3
```

At the top of your .py files.

Submitting jobs to ComputeCanada

You can run programs and executables from the terminal, but of course that takes up resources and defeats the purpose of having access to supercomputer clusters.

Your programs should be written in bash (.sh). To submit a job, you use

```
sbatch file_name.sh
```
To check on your submitted jobs, you can use

```
sq
```
Alternatively, you can use

```
squeue
```
to see all the jobs submitted by everyone. It is a lot.

Every job that runs produces a **slurm#######** file that contains the output of the job run.

Job Setup

In a bash file, you need to include headers to tell the system the amount of resources you require for your job. It is best practice to give your submitted job some extra resources (time/memory) beyond what you anticipate the job requires. If you exceed any of those resources before your job is submitted, your simulation is terminated. For example, you can use the heading like below -> change resources as required. The more conservative your resources, the faster your job will exit the queue and begin running, so a balance is required.

```
#!/bin/bash
#SBATCH --time=01:00:00
#SBATCH --mem=16G
#SBATCH --nodes=1
#SBATCH --cpus-per-task=32
#SBATCH --mail-user=your_email@ualberta.ca
#SBATCH --mail-type=ALL
```
So this is for a simple program. Gives 1 hour with 16 GB of memory. Uses 1 node with 32 cpus (32 cpus per 1 node, fyi). Put in your email to be notified when jobs start, finish, fail, or are cancelled. Prepare your inbox for a lot of emails.

I'll note here that to run a executable file, include in your program (or write in terminal), for example

```
./real.exe
```
Or
```
srun ./real.exe
```
To run python code, do

```
python your_python_code.py
```
I haven't used other languages on the clusters, but I am sure they follow a similar format.

Cancel Jobs

To cancel a job, you can use

```
scancel ############
```
where the ######## is the ID of the job you want to cancel. You can check that number with **sq**.

You can cancel all of your jobs using

```
scancel -u $USER
```
Alternatively, you can cancel all of your pending jobs using

```
scancel -t PENDING -u $USER
```

Programs

Be aware that as I describe the various programs, the lines in programs referenced in the document may be different to how they are in the programs themselves. I will do my best to be diligent and minimize errors/confusion. As I write up this document, I am trying a few new things. I hope that It does not get confusing beyond this.

Additionally, it is assumed that you have followed previous steps to link files and adjust the namelist.wps/namelist.input files appropriately.

As I have mentioned earlier that I installed multiple copies of the WPS and WRF software on different clusters. The programs and executables I wrote are enumerated, so that when I get email notifications, I can identify which programs were successful/failed.

Running WPS

The **WPSG1.sh** file exists inside the WPS directory, and simply runs geogrid, ungrib, and metgrid executables in order.

WPSG1.sh

```
#!/bin/bash
#SBATCH --time=01:00:00
#SBATCH --mem=16G
#SBATCH --nodes=1
#SBATCH --cpus-per-task=32
#SBATCH --mail-user=your_email@ualberta.ca
#SBATCH --mail-type=ALL
srun ./geogrid.exe
srun ./ungrib.exe
srun ./metgrid.exe
```

Running WRF

The **WRFG1.sh** file exists in the WRFV3/run directory, and simply runs real and wrf executables.

WRFG1.sh

```
#!/bin/bash
#SBATCH --time=16:00:00
#SBATCH --mem=125G
#SBATCH --nodes=2
#SBATCH --cpus-per-task=32
#SBATCH --mail-user=your_email@ualberta.ca
#SBATCH --mail-type=ALL
srun ./real.exe
srun ./wrf.exe
```

Attempts to automate WRF further.

I have a python code named **Automate_WRF.py**, to be placed in the **WRFV3/run** directory (I am not going to copy and paste it here, cause it is a bit lengthy). Since it has been recommended to run WRF in two-day bursts (allow for one day of data for the software to get to steady-state, take data from the second date), this means that you are running many individual simulations.

What **Automate_WRF.py** does is create the **namelist.wrf** files required for the simulations. These files are named as **_namelist_wrf_[date].txt**. Additionally, it creates a **_namelist_file.txt**, which stores all of the **_namelist_wrf_[date].txt** file names in reverse order.

In the **Automate_WRF.py**, it generates the text for the **namelist.wrf** file, which can then take that file's place and run the simulation. Running this program will overwrite the existing **namelist.wrf** file with the first one file generated.

```
python Automate_WRF.py
```

For adjustments, you will need to change the date range on **line 8 & 9** manually. As you can see, I am doing it in a week-long burst, for example. -> 4 WRF installations would be required to do a month, in this case, simultaneously.

```
start_threshold = datetime(2016, 1, 1)
end_threshold = datetime(2016, 1, 8)
```

For the rest of the file, you would need to change the **namelist.wrf** as required for your simulation, ensuring that parameters are aligned with your WPS **namelist.wps** file.

Then for submitting the jobs, you can use this variant of **WRFG1.sh.**

```
#!/bin/bash
#SBATCH --time=16:00:00
#SBATCH --mem=125G
#SBATCH --nodes=2
#SBATCH --cpus-per-task=32
#SBATCH --mail-user=your_email@ualberta.ca
#SBATCH --mail-type=ALL

line=$(head -
1 _namelist_file.txt ) # read the first line (filename) in _namelist_file.txt
echo $line
cp ${line} namelist.input # set the contents of that file to namelist.input
./real.exe  # run
./wrf.exe   # run
if [ -s "_namelist_file.txt" ] # If the file is not empty -> more sims to run
    echo "$(tail -
n +2 _namelist_file.txt)" > _namelist_file.txt # Remove the most recent file name
 from _namelist_file.txt
    sbatch WRFG* #Start the simulation again, to be done with the next file
fi
```

This job references the **_namelist_file.txt**, and replace the **namelist.wrf** with the first text file referenced in **_namelist_file.txt**. Simulation runs ./real.exe and ./wrf.exe. After running, it will remove the file that just ran from **_namelist_file.txt** and queue up a simulation for the next **namelist.wrf.**

Testing Domain Area

Before running many simulations, I recommend doing a quick 10-minute run to ensure that the domain spacing and location are adequate. After running a quick simulation, run the **Simple_Extract_Data.py** program. On line 11, you will need to specify which file you are reading; this will output to a .csv file in the same directory. From there, I recommend finding the MAX and MIN latitude/longitude values from the spreadsheet and throwing it into Google Maps to get an idea of the domain. Simply just finding MAX and MIN values is not exactly true of the boundaries of the domain, but it is close enough for the purposes of setting up your simulations.

Closest Latitude/Longitude

Using the Python code, **ClosetLatLong.py** can find the closest latitude and longitude locations to your points of interest from the NetCDF file. As you are basically placing a mesh/grid over an area to make your domain, the points in your simulations will not align with the actual locations you are interested in. But you can get close, or decent approximations.

In **line 10**, place the **(Latitude, Longitude)** coordinates you are interested in. In **line 28**, place the .csv file you wish to extract locations from – do only a small simulation, such as 10 minutes in duration, as suggested in the Testing Domain Area section. The output will give you the coordinates in the simulation that align closest to the specified points (in order). This step is not necessary, as it is superseded with **Extract_Data.py.**

Extracting Data from wrfout Files

There exist online applications for extracting information from NetCDF (.nc) files, which are the WRF simulations' output. NetCDF files are large multidimensional arrays that store all the data. For every time frame, for every latitude and longitude pair, it stores all the weather data variables. It is a lot. Extracting data can be done in Python.

Reference the program **Extract_Data.py**. In **line 10**, import the wrfout file to extract data from. In lines 14, place the **(Latitude, Longitude, Name -> (for saving purposes))** for the coordinates of interest.

Line 34 presents the **start_offset;** this is to control when in the time-series you want to begin extracting data. If you wanted to begin extracting at the start of the file, set it to 0. If you wanted to start a day into the simulation, for example, set it to 288 (if the simulation has a timestep of 5 minutes).

Line 39 presents the **end_offset;** this is to control when in the time-series you want to end extracting data. If you wanted to end extracting at the end of the file, set it to 0. If you wanted only to extract a day worth of data from the simulation, for example, set it to 288 (if the simulation has a timestep of 5 minutes).

Starting at **line 82** is where you extract your data for the variables desired. You can uncomment **line 11** or run **print(data.variables.keys())** to see the available parameters available. Variable extraction follows the format:

```
X_Wind = data.variables['U'][Time, Height Layer (if applicable), Latitude, Longitude]
```

Where 'U' is the X direction wind variable, in this example. You replace Time, Height Layer, Latitude, Longitude with respective numerical values. Using [:] selects all.

Line 87 sets up the columns for the CSV file. You need to add it, include units.

Starting on **Line 90** is where the program places the weather variables into the data frame. You should be able to follow the format presented to add new variables.

Line 97 saves the .csv file with the name given from your list starting on **Line 14**. I recommend saving the .csv files to a separate directory, such as **/Results/[location]/CSVFiles/[location]**

Variables that are noteworthy and may be worth extracting, though it depends on your application.

Variable Key	**Variable Name / Description**
XTIME or Times	Time parameter, in UTC time zone most likely
XLAT	Latitude
XLONG	Longitude
U10	X Direction wind at 10 meters elevation [m/s]
V10	Y Direction wind at 10 meters elevation [m/s]
T2	Temperature at 2 meters elevation [K]
Q2	Humidity at 2 meters elevation [kg/kg]
PSFC	Surface Pressure [Pa]
RAINC and RAINCC	Precipitation. Sum these two to get total precipitation [mm]
SWUPBC	Short Wave Radiation Flux Up [W/m^2]
SWDNBC	Short Wave Radiation Flux Down [W/m^2]
LWUPBC	Long Wave Radiation Flux Up [W/m^2]
LWDNBC	Long Wave Radiation Flux Down [W/m^2]
SWDOWN	Sunshine duration [s]
ACSWDNB	[Accumulative] Downward Short Wave Surface Radiation. [W/m^2]. *Starts from the largest value and gets smaller -> Do first value – that timestep value to correct.
ACLWDNB	[Accumulative] Downward Long Wave Wave Surface Radiation. [W/m^2] *Starts from the largest value and gets smaller -> Do first value – that timestep value to correct.

Concatenate CSV

Use **ConcatCSVFiles.py** to merge CSV files. My directory structure may be different from yours, so be aware of file location and directory. The python file's comments should describe what is going on pretty well; effectively, it takes whatever files you throw at it and combine them into one .csv file, organized by time. It also outputs results to 3 significant digits, which you can change if you so desire.

You will need to change the directories' names, filename and save locations to fit your data/structure. Change file names/directories on **Line 13**. There are 3 locations to change working directories on **lines 34**, **35, 40.** And you can change the file save name on **Line 47.**

I recommend saving the .csv files to a separate directory, such as
/Results/[location]/ConcatCSVFiles/[location]

Fix Data and Clean Series

Last thing to do is check for missing data and clean some erroneous data, if applicable. This is done using **FindMissingData.py**.

Fixing Data

For example, if you are getting radiation flux parameters, the first output from a simulation is 0 W/m2. So part of the simulation attempts to fix that. You will need to look at your output files to see if you have any outstanding issues, such as the one I described. Probably the best method is to either know the condition upon which the data is wrong (such as 0 W/m2 solar radiation) or if the error repeats on a regular interval (beginning of every simulation/day).

So set the bad data to null and interpolate to fix it. The first row of data can be amended using backfill, as you cannot interpolate the first data point. It is a design choice to fix the data.

Missing Data

The second part of the program goes through the time series of the .csv file and checks to see if the output is of the same interval. You will need to change the time difference you are expecting on **Line 59**. In the current case, I was looking at a 5-minute difference. If there is an error, it will print the time of the issue and the index line so that you can check afterwards in excel.
If an error is detected, you will likely need to check your simulations, make sure that time is simulated, extract the data, concatenate, and check again to fix it.

Final Comments and Closing

Congrats, you've made it to the end. I hope this helps and you can avoid the many errors and issues that I came across. If you have questions, you can email me at nwoodhou@ualberta.ca. Updates may be available on my GitHub: https://github.com/NigelWoodhouse

Depending on how long you are reading this after this is posted, I may or may not see it / remember what I did.

I hope this helps, and best of luck.

Links and Additional Resources

Compute Canada Tutorial for using WPS/WRF on their system. Includes recompiling WPS/WRF.

https://wiki.math.uwaterloo.ca/fluidswiki/index.php?title=WRF_Tutorial

Useful forum for WPS and WRF errors

https://forum.mmm.ucar.edu/phpBB3/viewforum.php?f=3&sid=35a12661f754ae59f8b21ed0ac723cf9

Installing Python on Clusters

https://docs.computecanada.ca/wiki/Python

Best practices for WPS/WRF domain creation

https://www2.mmm.ucar.edu/wrf/users/namelist_best_prac_wps.html

Extracting NetCDF Data with Python

https://www.youtube.com/watch?v=hrm5RmsVXo0

WRF Documentation and Variables

https://www2.mmm.ucar.edu/wrf/users/docs/user_guide_v4/v4.0/users_guide_chap5.html

Programs from my GitHub

https://github.com/NigelWoodhouse