NCAS Unified Model Introduction: Practical sessions (Rose/Cylc)

NCAS-CMS

18 March 2021

CONTENTS

Practical exercises for the UM training course in Reading, 6-8 November 2019.

NCAS Computational Modelling Services: http://cms.ncas.ac.uk/

SECTION

ONE

GETTING SET UP

1.1 Install and Launch MobaXterm

Unfortunately we can no longer pre-install MobaXterm for you due to licencing restrictions, so you will first need to download and install MobaXterm:

- Login to the windows machine with your University of Reading credentials
- From Chrome, go to page: https://mobaxterm.mobatek.net/download.html
  Google "mobaxterm download" and this should be the first entry (check the URL).
- Under "Home Edition" select "Download now"
- On next page select **"MobaXterm Home Edition v12.3 (Portable edition)"**.
  This should download the package.
- Click the download icon in the bottom left hand corner.
- Double-click on the **MobaXterm_Personal_12.3** application file, and select "Extract all".
  A new directory window will open up.
- Double-click **MobaXterm_Personal_12.3** to launch the application.

Next time, navigate to "Downloads" to open the application.

1.2 Login to PUMA

```
xterm$ ssh -Y <puma-username>@puma.nerc.ac.uk
```

1.3 Set up your PUMA environment and access to MOSRS

i. Configure ~/.profile

If this is the first time you have used your PUMA account, you will need to create a `.profile`. Copy our standard one:

```
puma$ cd
puma$ cp ~um/um-training/setup/.profile .
```

(If you already have a `.profile`, make sure it includes the lines from the standard file.)

ii. Configuring access to MOSRS

Run the `mosrs-setup` script which will take you through the set up process to access the Met Office Science Repository Service (Remember your MOSRS username is one word; usually firstnamelastname, all in lowercase):

```
puma$ ~um/um-training/mosrs-setup
```

Log out of PUMA and back in again (you will get a warning about not being able to find `~/.ssh/ssh-setup` this can be ignored and will be resolved in the next step). You should be prompted for your Met Office Science Repository Service password. A new window should then pop up (it may be hidden behind other windows) for `Rosie` asking for **Username for 'u' - 'https://code.metoffice.gov.uk/rosie/u'** . Enter your MOSRS username again.

Note: The cached password is configured to expire after 12 hours. Simply run the command `mosrs-cache-password` to re-cache it if this happens. Also if you know you won't need access to the repositories during a login session then just press return when asked for your MOSRS password.

1.4 Make sure you can login to your ARCHER training account

An ARCHER username will be provided on the day of the course. Ask the CMS team if you're unsure. The password for these accounts is listed in a file on PUMA: `~um/um-training/login.txt`.

From PUMA:

```
puma$ ssh <archer-username>@login.archer.ac.uk
```

Note: It's best to copy and paste the password from the file rather than type it by hand.

1.5 Set up your ARCHER environment

Once you have successfully logged into ARCHER, copy the following profile to your home directory.

```
archer$ cp /work/y07/y07/umshared/um-training/rose-profile ~/.profile
```

1.6 Exit ARCHER

Log out of ARCHER. This should take you back to puma.

```
archer$ exit
```

1.7 Set up an ssh connection from PUMA to ARCHER

Note: If you already have ssh keys and agent set up on PUMA, follow the instructions on *Using an existing ssh agent* in the Appendix, then skip to Section 1.8.

i. Generate authentication key on PUMA and install it on ARCHER.

Run the `install-ssh-keys` script. This will take you through ssh-key creation and copy the key over to ARCHER.

```
puma$ source ~um/um-training/install-ssh-keys <archer-username>@login.archer.
↪ac.uk
```

When prompted to **Enter passphrase**, this should be a fairly complicated and unguessable passphrase. You can use spaces in the passphrase if it helps you to remember it more readily. It is recommended that you don't use your password in case it is hacked.

Warning: DO NOT use an empty passphrase as this presents a security issue.

After generating your ssh-key, the script will copy it over to ARCHER.

When prompted for **Password**, enter your ARCHER password.

ii. Verify the authentication works.

```
puma$ ssh <archer-username>@login.archer.ac.uk
Enter passphrase for key '/home/<puma-username>/.ssh/id_dsa':
[TYPE_YOUR_PASSPHRASE]
```

If you don't get asked for your Passphrase (i.e. RSA key), then something has gone wrong. In this case, make sure the public key, was successfully copied over to ARCHER by logging into ARCHER and opening the file `~/.ssh/authorized_keys`. It should contain something similar to:

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAt1JmHYgsuf0UWVLqNqnDSaUUP2xJ+Um0H5WnUt
/i2mxhlBrwOtvVWRjnzo5EcylZJs/Cg5JVe4UR6toqNXbZG1RXscLQnQoPAvzFoWLzfP7Q3lrz
eC1SkM2FWfWC38ga3Svs6fm63/I7WmJy+4D8BWWaXj/9yM1OskFj6yfWItr150rwwNauOQbWJh
l7I/KkfhVPBvZ9vHiAK4cjUMQ9fFS1dij3GSBmOfu2RuMgNNg9y1MLSzEk2242F4tOg7paTk7w
wUZ+ZLqRBtT2aREnjIGI7KvACBZD1y40tXXPIZw9m2Dl0dK7mFQ2/YFWh2/NAmkFMXzDOmkg0b
iq1m+QKw==
ros@puma
```

If it doesn't, and no errors were reported from the `install-ssh-keys` script, please ask for assistance.

Once you have this part working, log out of ARCHER.

iii. Start up ssh-agent.

Run the following command and type your passphrase:

```
puma$ ssh-add
Enter passphrase for /home/<puma-username>/.ssh/id_rsa:
[TYPE_YOUR_PASSPHRASE]
```

The ssh agent should keep running even when you log out of puma, however you may need to restart it from time to time. For instructions on how to do this see *Restarting your ssh agent* in the Appendix.

1.8 Check this all works by ssh-ing to ARCHER

From PUMA type:

```
puma$ ssh <archer-username>@login.archer.ac.uk
```

If you get to ARCHER without a password or passphrase, then you're done.

You are now ready to try running a UM suite!

SECTION

TWO

WORKING WITH SUITES

2.1 Suite Discovery and Management: rosie go

Rosie go is the suite manager GUI. It acts as a hub for all your suite work. From `rosie go` it is possible to search for suites, create, checkout, delete, edit, run suites and more. We will take a look at the main features here.

Launch `rosie go` by typing:

```
puma$ rosie go
```

i. Searching for suites

By default, `rosie go` will show all the suites that you have checked out locally, so unless you have used Rose before you will have an empty results panel to begin with. You can search for suites by typing a word/phrase into the search box and either click the search button or press `<Enter>`. The search looks for the entered word/phrase in **any** of a suite's properties.

- Enter **u-ag137** in the *Search* box and press `<Enter>`

You should now see a long list of suites. All these suites are listed as they reference `u-ag137` in their suite information.

More advanced queries can be run by clicking the **+** button next to the search button. Queries allow you to filter results based on the values of particular properties. You can combine filters to make complex queries.

- Select **idx** in the property box and **eq** in the operator box and then enter the value **u-ag137**. Click **Query**.

This time you should only see one suite listed in the results pane.

- Now run a query to list all suites **owned by rosalynhatcher containing ARCHER in their title**.

The searches you run within `rosie go` are recorded in your search history.

- View your history by clicking *"History -> Show search history"*.

The Search History panel should now be displayed on the left-hand side of `rosie go` listing your past searches. You can order the results by type, parameters or whether you asked to see all revisions by clicking on the relevent column head. To re-run one of these searches simply click on it.

- Try running your initial search for u-ag137 again.

Close the History Panel by clicking the close button at the top of the history panel.

ii. Viewing suite information

To obtain more information about a suite listed in the search results you can do one of two things; hold your mouse over the suite to display a tooltip containing more details or right click on the suite and click `Info` in the pop-up menu to display a dialog box containing further details.

- What project is suite u-ab878 associated with?

Suite search results can be ordered by property in either ascending or descending order. To do so, click on the column title for the property you wish to order by so an arrow is displayed next to it indicating the order in which the property is being sorted.

iii. Checking out an existing suite

Right-clicking on a suite displays a pop-up menu from which you can perform many functions on the suite; e.g. checkout, copy, delete, run, etc. We will perform many of these actions in subsequent exercises but to begin with we will just checkout an existing suite to use in the *"Editing Suites"* exercises. To view a suite it must be checked out first.

- Right-click on suite **u-ag137** and select *Checkout Suite* from the pop-up menu.

When you checkout a suite it is always placed in your `~/roses` directory. In this state, the suite is simply a working copy - you can edit it and run it but any changes you make will only be held locally.

Note: You can also checkout a suite by highlighting it and then clicking the *Checkout* button on the toolbar.

iv. Other useful features

To see what suites you have checked out click the `Show local suites` button to the left of the search box (represented by the *home* icon). You should have at least 1 suite listed.

- What do you think the *house* icon in the local column indicates?

2.2 Editing Suites: rose edit

The Rose config editor in combination with the meta-data file, which describes UM inputs, is the GUI for editing UM suites. Building and running the UM under Rose requires, at least, two separate apps: an fcm_make_app to build the model executable and a UM app to configure the runtime namelists and environment variables. Coupled models may require additional fcm_make apps, one for each executable to be built.

i. Launch the config editor GUI

Right click on suite u-ag137 and select `Edit Suite`. The `rose edit` GUI will start up.

On the left hand side is a navigation panel containing a tree listing the apps in the suite. For this particular suite these are:

- *suite conf* - General suite configuration options
- *fcm_make_pp* - Extract and build the post-processing scripts
- *fcm_make_um* - Extract and build the UM source code
- *housekeeping* - Tidies up log files, old work and data directories
- *install_ancil* - Install ancillary files
- *postproc* - Post-processing settings

- *rose_ana* - Rose built in app; used here for comparison of dump files
- *rose_arch* - Rose built in app; used here for archiving of log files
- *um* - The UM atmosphere and reconfiguration settings

ii. Explore the GUI

Click on the triangle to the left of *suite conf* to expand that section. Click on *Build and run switches*. A panel will appear on the right-hand side containing options for controlling what tasks will be run for this suite. You can see that it will build the UM and reconfiguration executables, run the reconfiguration and then run the model.

Note: We generally use a **common notation** to help users navigate through the GUI and to help us help you with questions. Getting to "UM Science Settings" would be indicated like this: *um->namelist->UM Science Settings*. This notation will be used throughout the rest of this document.

The input namelists for the um are contained in the *um->namelist* section. Let's take a look at the science namelist for Microphysics (Large Scale Precipitation), run_precip under "UM Science Settings".

For each UM namelist item there is a short description to help you understand what that variable is. Click on the cog next to a namelist variable and select *Help* to view more detailed information. The help information can give you some useful pointers but be aware that it has been written with Met Office setup in mind.

Range and type checking of variables is done as soon as the user enters a new value. Try changing the value of *timestep_mp_in* to 0. This will cause an error flag to appear, hover over the error for more information and click the *undo* button several times to revert to the original value.

Some larger science sections have been been divided into subsections, take a look at "Section 05 - Convection" for an example of this. To open a section in a new tab click with the middle mouse button, expand the section by clicking the page triangles. Rose edit has a search box which can be used to search item names. Try searching for the variable *astart* where the input dump is specified, you will be taken directly to the Dumping and Meaning panel.

Trigger ignored settings are hidden by default and only appear to the user when the appropriate options are selected. Open the Gravity Wave Drag panel, if you change *i_gwd_vn* from 5 to 4 the options available change. Click the save button to apply these changes to your app. Let's take a look at what effect this has had to the `rose-app.conf` file, run `fcm diff` in the suite directory.

```
puma$ cd ~/roses/u-ag137
puma$ fcm diff -g
```

You should see that several namelist items have had *!!* added to the start of the line. This tells Rose to ignore these items when processing the app file into Fortran namelists. Should you wish to see all variables on a panel select "View All Ignored Variables" and "View Latent Variables" from the "View" menu.

Switch back to the Rose edit window and click the **undo** button to revert the changes and then **save** the suite again. To view all changes made to the suite in the current session click on the **Undo/Redo Viewer** in the *Edit* menu.

iii. Error checking of UM inputs

In addition to the type and range checking of namelist items and environment variables, more thorough checks can be made using Rose macros and the fail-if/warn-if metadata.

First let's check if the suite contains any options which trigger the fail-if and warn-if checks in the UM metadata. Select “Check fail-if, warn-if” from the “Metadata” menu. As this suite is setup correctly “FailureRuleChecker: No problems found” should appear at the bottom right of the window.

Now let’s try and introduce both a warning and a failure. We’re going to change the boundary layer option “alpha_cd”. Either navigate to *“Section 03 - Boundary Layer -> Implicit solver options”* or type “alpha_cd” into the search bar. Click on the plus sign to add an array element to alpha_cd and type 1.5 into the new box. Next navigate to *“Reconfiguration and Ancillary Control -> Output dump grid sizes and levels”* and increase the number of ozone levels to 86. Now run the fail-if, warn-if checker again.

- What is the error?
- What is the warning?

Use the undo button to put the settings back to how we found them and run the checker again. It is strongly recommended that whenever namelists and environment variables are modified that the fail-if, warn-if checker is applied before running the suite.

SECTION

THREE

RUNNING A UM SUITE ON ARCHER2

3.1 ARCHER2 architecture

In common with many HPC systems, ARCHE2 consists of different types of processor nodes:

- **Login nodes:** This is where you land when you ssh into ARCHER2. Typically these processors are used for file management tasks.
- **Compute / batch nodes:** These make up most of the ARCHER2 system, and this is where the model runs.
- **Serial / post-processing nodes:** This is where less intensive tasks such as compilation and archiving take place.

ARCHER2 has two file systems:

- **/home:** This is relatively small and is only backed up for disaster recovery.
- **/work:** This is much larger, but is not backed up. Note that the batch nodes can only see the work file system. It is optimised for parallel I/O and large files.

Consult the ARCHER2 website for more information: http://www.archer2.ac.uk

3.2 Running a Standard Suite

To demonstrate how to run the UM through Rose we will start by running a standard N48 suite at UM11.8.

i. Copy the suite

- In `rosie go` locate the suite with idx **u-cc519** owned by **rosalynhatcher**.
- Right click on the suite and select `Copy Suite`.

This copies an existing suite to a new suite id. The new suite will be owned by you. During the copy process a wizard will launch for you to edit the suite discovery information, if you wish.

A new suite will be created in the MOSRS rosie-u suite repository and will be checked out into your `~/roses` directory.

ii. Edit the suite

- Open your new suite in the Rose config editor GUI.

Before you can run the suite you need to change the *userid*, *queue* and *account code*.

Now make the following changes:

- Click on *suite conf* -> *jinja2* in the left hand panel
- Change HPC_USER (that's your ARCHER2 training account)
- Change HPC_ACCOUNT to **'n02-training'**
- Change HPC_QUEUE to be the reservation code for today. (e.g. **'R6585903'**)
- Save the suite (*File* -> *Save* or click the *down arrow* icon)

Note: Quotes around the 'n02-training', reservation code and your ARCHER2 username are essential otherwise the suite won't run.

Note: In normal practice you submit your suites to the parallel queue (either 'short' or 'standard') on ARCHER2. For this training course, we are using processor Reservations, whereby we have exclusive access to a prearranged amount of ARCHER2 resource. Reservations are specified by a reservation code; e.g. R5212096.

iii. Run the suite

The standard suite will build, reconfigure and run the UM.

- Click on the triangle symbol on the right end of the menu bar to run the suite.

Doing this will execute the `rose suite-run` command (more on this later) and start the Cylc GUI (gcylc) through which you can monitor the progress of your suite graphically. The cylc GUI will update as the job progresses.

iv. Looking at the queues on ARCHER2

While you're waiting for the suite to run, let's log into ARCHER2 and learn how to look at the ARCHER2 queues. Remember you will need to login to ARCHER2 from your local desktop NOT from PUMA.

Run the following command:

```
squeue -u <archer2-user-name>
```

This will show the status of jobs you are running. You will see output similar to the following:

```
ARCHER2> squeue -u ros
      JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
     148599  standard u-cc519.      ros  R       0:11      1 nid001001
```

At this stage you will probably only have a job running or waiting to run in the serial queue. Running `squeue` will show all jobs currently on ARCHER2, most of which will be in the parallel queues.

Another useful command is `serialJobs`, which lists the jobs in the serial queue only. You will need to run `module load anaconda` before running the `serialJobs` command. Try it now:

```
ncastr01@eslogin005:~> module load anaconda
ncastr01@eslogin005:~> serialJobs
```

Once your suite has finished running the Cylc GUI will go blank and you should get a message in the bottom left hand corner saying *'Stopped with succeeded'*.

Cylc is set up so that it *polls* ARCHER2 to check the status of the task, every 5 minutes. This means that there could be a maximum of 5 minutes delay between the task finishing on ARCHER2 and the Cylc GUI being updated. If you see that the task has finished running but Cylc hasn't updated then you can manually poll the task by right-clicking on it and selecting **Poll** from the pop-up menu.

3.3 Standard Suite Output

The output from a standard suite goes to a variety of places, depending on the type of the file. On ARCHER2 you will find all the output from your run under the directory `~/cylc-run/<suitename>`, where `<suitename>` is the name of the suite. This is actually a symbolic link to the equivalent location in your `/work` directory (E.g. `/work/n02/n02/<username>/cylc-run/<suitename>`.

Rose bush

The standard output and errors from the suite can be easily viewed using Rose Bush.

For suites submitted from PUMA; in a browser navigate to: http://puma.nerc.ac.uk/rose-bush

Enter your PUMA userid and click *"Suites List"*. You should then see a list of all the suites you have run. Click on *"tasks jobs list"* for the suite you have just run. You can examine the output of each task using the links, as well as see whether the suite contains failed tasks, or is currently running. For this suite you should see output files for 4 tasks: fcm_make (code extraction), fcm_make2 (compilation), recon & atmos. The job.out and job.err files are the first places you should look for information when tasks fail.

Note: To run Rose Bush on Monsoon run: `firefox http://localhost/rose-bush`

Compilation output

The output from the compilation is stored on the host upon which the compilation was performed. The output from fcm_make is inside the directory containing the build, which is inside the *share* subdirectory.

```
~/cylc-run/<suitename>/share/fcm_make/fcm-make2.log
```

If you come across the word "failed", chances are your model didn't build correctly and this file is where you'd search for reasons why.

Standard output

The output from the UM scripts and the output from PE0 is placed in the `log` subdirectory. As we saw in Rose Bush stdout and stderr are written to 2 separate files. For a task named *atmos*, the output from the most recent run will be:

```
~/cylc-run/<suitename>/log/job/1/atmos/NN/job.out
```

And the corresponding error file is:

```
~/cylc-run/<suitename>/log/job/1/atmos/NN/job.err
```

Here `NN` is a symbolic link created by Rose pointing to the output of the most recently run *atmos* task.

Take a look at the `job.out` for the *atmos* task either on the command-line or through Rose Bush.

- How much walltime did the run consume? Hint: go to the bottom of the file and find walltime.
- Why does the phrase walltime appear twice?
- How much time did you request for the task?

- How many AUs (Accounting Units) did the job cost? Hint: 1 core hour currently = 15 AUs (You should have some idea of the resource requirements for your runs and how that relates to the annual AU budget for your project). See the ARCHER website for information about the AU.
- Did the linear solve for the Helmholtz problem converge in the final timestep?
- How many prognostic fields were read from the start file?

Binary output - work and share

By default the UM will write all output to the directory it was launched from, which will be the task's `work` directory. However, all output paths can be configured in the GUI and in practice most UM tasks will send output to one or both of the suite's `work` or `share` directories.

```
~/cylc-run/<suitename>/work/1/atmos
```

or

```
~/cylc-run/<suitename>/share/data
```

For this suite output is sent to the `work` directory.

Change directory to the work space.

- What files and directories are present?

Model diagnostic output files will appear here, along with a directory called `pe_output`. This contains one file for each processor, for both model and reconfiguration, which contain logging information on how the model behaved.

Open one of these files `<suite-id>.fort6.peXX` in your favourite editor.

The amount of output created by the suite and written to this file can be controlled in the suite configuration (*um -> env -> Runtime Controls -> Atmosphere only*). For development work, and to gain familiarity with the system, make sure "Extra diagnostic messages" are output. Switch it on in this suite if it isn't already.

It is well worth taking a little time to look through this file and to recognise some of the key phrases output by the model. You will soon learn what to search for to tell you if the model ran successfully or not. Unfortunately, important information can be dotted about in the file, so just examining the first or last few lines may not be sufficient to find out why the model hasn't behaved as you expected. Try to find answers to the following:

- How many boundary layer levels did you run with?
- What was the range of gridpoints handled by this processor?

Check the file sizes of the different file types. The output directory will contain start dumps, diagnostic output files and possibly a core dump file if the model failed) and these usually have very different sizes.

SECTION

FOUR

FCM TUTORIAL

In this section you will learn about:

- Incorporating modifications to your suite
- Creating a ticket to document your code change
- Creating a branch using the FCM GUI and command line
- How to checkout and change a working copy
- Getting information about your working copy and branch
- Committing changes to your branch
- Version control of suites

4.1 Setting up your default text editor

When you attempt to create a branch or commit changes to the repository, you will normally be prompted to edit your commit log message using a text editor. The system chooses its editor by searching for a non-empty string through a hierarchy of environment variables in this order: SVN_EDITOR, VISUAL, and EDITOR. Note that the editor you select must able to run in the foreground. The default editor is *vi*. If you prefer to use *emacs* or *nedit*, for example, you can add one of the following in your `$HOME/.profile`, `$HOME/.kshrc` (Korn) or `$HOME/.bashrc` (Bash):

```
# Emacs
export SVN_EDITOR=emacs

# NEdit
export SVN_EDITOR='nedit'
```

4.2 Applying modifications to a UM suite

Now that you've run a basic suite, you need to know how to apply changes to the trunk code. We'll start by applying existing changes (aka a branch) to your suite. A branch equates to a copy of the UM source code tree with user modifications. A branch can contain changes to many code files.

As a user, you will be supplied with the **URL** and possibly a version number of a branch. FCM keywords can be used to specify the URL which is a way of using shorthand for standard bits of the URL path. For example, this branch has been created in the MOSRS UM repository:

```
https://code.metoffice.gov.uk/um/main/branches/dev/rosalynhatcher/vn10.5_um_
↪shell1
```

A shorter way to specify this is:

```
fcm:um.x-br/dev/rosalynhatcher/vn10.5_um_shell1
```

This branch contains a code change to `um_shell` to print:

`Tutorial Change:  Start of UM RUN Job:` instead of `Start of UM RUN job:`

To add this branch go to the panel *fcm_make* -> *env* -> *sources* and in the **um_sources** box add the URL:

```
branches/dev/rosalynhatcher/vn10.5_um_shell1
```

Now **Save** your suite and then submit it either by running `rose suite-run` or clicking the "Play" button in the rose edit GUI.

When the suite has completed, check the `job.out` file to verify that you have got the changed output message:

`Tutorial Change:  Start of UM RUN Job:` instead of `Start of UM RUN job:`

4.3 Opening a Ticket

All code changes destined for the UM trunk must have an associated Trac Ticket documenting the change (e.g. https://code.metoffice.gov.uk/trac/um/ticket/1957). For all other changes creating a ticket to document your change is entirely optional.

Note: For full details on the UM working practices including the review and commit to trunk process please see https://code.metoffice.gov.uk/trac/um/wiki/working_practices. If you are planning to make a code change intended for the trunk please make sure you read this before starting the change.

We will now create a ticket for our code change. Navigate to the UM Trac page on MOSRS: https://code.metoffice.gov.uk/trac/um (You will be prompted to login to MOSRS if you haven't already done so). Click on **New Ticket**. A form entitled **Create New Ticket** should appear. For the purposes of this tutorial fill in the following:

- **Summary**: *"Code change for UM Tutorial"*
- **Description**: provide as much detail as possible so that all reading the ticket are able to understand the planned change. You can use WikiFormatting to enhance the readability of your text.
- **Type**: select *"task"*
- **Milestone**: select *"Not for Builds"*; this indicates the ticket is not intended for the UM trunk. If your change is intended for the trunk this should be set to the UM release the change is being targeted at: e.g. *"UM10.6 code release"*
- **Severity**: leave this as *"minor"*
- **Component**: select *"General"*
- **cc**: leave this empty

- **Owner**: assign the ticket to yourself by selecting your username from the drop down list.

A preview of your ticket should appear at the bottom of the page. If if doesn't appear automatically, use the **Preview** button to inspect your ticket before using the **Create ticket** button to create the ticket.

Remember the number of your new ticket as you will need it later in this tutorial.

4.4 Making Code Changes

The default text editor for entering commit messages is `vi`. If you would prefer to use a different editor; for example `emacs` or `vim`, please see the section on "Setting up your default text editor".

i. Creating a branch

Firstly create a new directory (e.g. `um/branches`) in your `$HOME` directory on PUMA which will be your work area and `cd` to it.

Create a new branch by running the command:

```
fcm branch-create -k <ticket> <branch_name> fcm:um.x-tr@vn10.5
```

Where:

- **<ticket>** - is the related Trac ticket number for the ticket you created earlier.
- **<branch_name>** - is a short name for the branch. This must contain only alpha-numeric characters and/or underscores; e.g *tutorial*

You will be prompted to edit the message log file. A standard template is automatically supplied and pops up in your default text editor. Add a comment about what the branch is for at the top of the file. When you are ready, save your change and exit the editor. Answer **y** when you are prompted to go ahead and create the branch.

If the branch is created successfully you will get a message similar to the following:

```
Committed revision 52466.
[info] Created: https://code.metoffice.gov.uk/svn/um/main/branches/dev/
↪rosalynhatcher/vn10.5_tutorial
```

The branch will have a URL (location in repository) like this:

`https://code.metoffice.gov.uk/um/main/branches/dev/[userid]/vn10.5_[branch_name]`

By default FCM prepends the revision of the trunk you have branched from to your branch name. Here, as we have used version labelling it is **vn10.5**. If you had entered a version number instead of a label FCM would have added rxxx where xxx is the revision number instead.

Note: For further information on the options available for branch creation type: `fcm branch-create --help`

Take a note of the revision number the branch was created at, and the branch name, vn10.5_[branch_name].

You can see your branch from within the MOSRS Trac (https://code.metoffice.gov.uk/trac/um): Click on **Browse Source** on the Trac menu bar and then navigate through:

main–>branches–>dev–>[userid]

Your branch will also appear on the UM repository mirror held on PUMA (within 5 minutes): https://puma.nerc.ac.uk/trac/um.xm

ii. Making changes to a working copy

Checking out a working copy

You may have noticed that creating a branch does not create a source code tree that you can edit (working copy)! To do this you need to *Checkout* your branch. Make sure you have changed to the working directory you created earlier as by default code is checked out to the current directory. To checkout a copy of the UM code type:

```
fcm checkout URL
```

Where URL is the url of your branch. This can be supplied in it's full form:

```
https://code.metoffice.gov.uk/svn/um/main/branches/dev/[userid]/vn10.
5_[branch_name]
```

or by a shorter way:

```
fcm:um.x-br/dev/[userid]/vn10.5_[branch_name]
```

Note:

- In the second form we have replaced the leading part of the Subversion URL `https://code.metoffice.gov.uk/um/main/branches` with the FCM repository keyword `fcm:um.x-br`. Keywords are shortcuts to save you from having to type in the full URL.
- As we have not specified a local directory PATH in the checkout command, it will create a working copy in your current working directory, using the basename of the URL you are checking out. For example, when you checkout the branch you have just created, the command should create the working copy in `$PWD/vn10.5_[branch_name]`. Make a note of the location of your working copy, in case you forget where you have put it.
- We are also not specifying a revision to checkout, so it will checkout the HEAD, i.e. the latest revision.

Changing code

Back in the work area directory you created at the beginning of branch creation you should now see that a new directory has appeared and that it is named the same as your branch. This is your *working copy*. cd into this directory and explore the code structure to familiarise yourself with how the code is structured.

Now make some code changes! Use the following scenario to take you through the basic method of changing, adding and deleting files:

- Change to the `src/control/top_level/` sub-directory in your working copy.
- Edit `um_shell.F90`, using your favourite editor
- Go to the line that says **CALL umPrint('I am PE '//TRIM(str(mype))//' on '//TRIM(env_myhost),**
- Change: 'I am PE' to 'Hello World PE'
- Go to the line that says **of UM RUN Job :**
- Change: 'of UM RUN Job :' to 'of UM Tutorial RUN Job:'
- Save your changes and exit the editor

Adding a new file

- Still in the `src/control/top_level` directory, add a new FORTRAN module file (`um_shell_mod.F90`) containing a subroutine called `um_shell_sub()`. (An example file is available on PUMA: `~um/um-training/um_shell_mod.F90`. The routine `umPrint` should be used for writing out messages rather than standard FORTRAN `WRITE` statements.)
- Run **fcm add** on the command line, to let the repository know you're adding a new file at the next commit. Make sure you are still in `src/control/top_level` and then type:

  ```
  fcm add um_shell_mod.F90
  ```

  at the command prompt.
- Modify `um_shell.F90` to use this new module. You'll see lots of `USE` statements near the top of the file. Add the following to use our new one.

  ```
  USE um_shell_mod
  ```

- Then add a line to call the `um_shell_sub` subroutine (suggest around line 375):

  ```
  CALL um_shell_sub()
  ```

Deleting a file

- In the `fcm-make/ncas-xc30-ifort` directory, you should see a file `um-createbc-safe.cfg`
- Run **fcm delete** on the command line, to let the repository know you want to remove this file from your branch: Make sure you are in `fcm-make/ncas-xc30-ifort` and then type:

  ```
  fcm delete um-createbc-safe.cfg
  ```

Getting information about changes to a working copy

All the changes you have made so far have not been committed - i.e. saved to your branch in the repository. It is possible to list these changes using the `fcm status` command. Firstly, make sure you cd back up to the top level of your working directory and then type:

```
fcm status
```

and you should see a list of files that have been changed. If you've followed the example scenario above you should see output similar to this:

```
ros@puma$ fcm status
D       fcm-make/ncas-xc30-ifort/um-createbc-safe.cfg
M       src/control/top_level/um_shell.F90
A       src/control/top_level/um_shell_sub.F90
```

Notice that each changed file is flagged with a letter that indicates what the change was: **A** for Added, **D** for Deleted and **M** for Modified.

Reverting an uncommitted change

At this point you can undo any changes before committing. Try the following so that you know how to restore a changed file:

- Edit `src/control/top_level/initial.F90` to make any change and then save it.
- Run `fcm status` again to confirm it has been flagged as Modified.

- Run `fcm revert` on the command line: Make sure you are still in `src/control/top_level` and then type `fcm revert initial.F90`
- Re-run `fcm status` to see that the file is no longer modified.

Note that **revert** will undo ALL changes to a file relative to your branch. Therefore if you've made several uncommitted changes, **revert** will undo them all, not just the last one.

iii. Committing changes

The change in your working copy remains local until you commit it to the repository where it becomes permanent. If you are planning to make a large number of changes, you are encouraged to commit regularly to your branch at appropriate intervals. Make sure you are in the top level directory of the working copy and then type:

```
puma$ fcm commit
```

A text editor will appear to allow you to edit the commit message. You must add a commit message to describe your change above the line that says **"–Add your commit message ABOVE - do not alter this line or those below–"**. Your commit will fail if you do not enter a commit message. Make sure you provide meaningful commit messages (if your change is intended for inclusion in the trunk you should reference your ticket number) as these will show up in the revision logs and can be a useful source of informtion.

DO:

- Put a link to the ticket that raises the issues you are addressing using a wiki syntax; e.g. #15. Putting this as the first item in the commit message means it will show very clearly under Trac what ticket the change relates to.
- State the reason for the change
- List possible impacts to other users
- Use wiki syntax that can be displayed nicely in plain text

DON'T:

- Repeat what's already stated in the merge template; e.g. statements such as *'merge my branch to the trunk'* should be avoided
- List the files you have changed. This will already have been included in the commit log by FCM
- Use wiki syntax that cannot be displayed nicely in plain text
- Be vague. A commit message that just says *"Fix"* is insufficient!

Save your change and exit the editor. Answer **y** when you are prompted to confirm the commit.

If you've followed the example scenario above you should see output similar to this:

```
ros@puma$ fcm commit
[info] emacs: starting commit message editor...
Change summary:
--------------------------------------------------------------------------
[Root    : https://code.metoffice.gov.uk/svn/um]
[Project: main]
[Branch : branches/dev/rosalynhatcher/vn10.5_example_branch]
[Sub-dir: ]

D       fcm-make/ncas-xc30-ifort/um-createbc-safe.cfg
```

(continues on next page)

(continued from previous page)

```
M       src/control/top_level/um_shell.F90
A       src/control/top_level/um_shell_sub.F90
----------------------------------------------------------------------
Commit message is as follows:
----------------------------------------------------------------------
#2412 - Testing Code Changes section of the tutorial
----------------------------------------------------------------------
Would you like to commit this change?
Enter "y" or "n" (or just press <return> for "n"): y
Deleting       fcm-make/ncas-xc30-ifort/um-createbc-safe.cfg
Sending        src/control/top_level/um_shell.F90
Adding         src/control/top_level/um_shell_sub.F90
Transmitting file data ..
Committed revision 29416.
Updating '.':
At revision 29416.
```

iv. Getting information about your branch

If you need to find out information about your (or other users') branches, you can use the **fcm branch info** command.

In the directory where you checked out the code, type:

```
puma$ fcm branch-info
```

You should see information about your branch revision, when it was last changed and the parent it was created from:

```
ros@puma$ fcm branch-info
URL: https://code.metoffice.gov.uk/svn/um/main/branches/dev/rosalynhatcher/
vn10.5_example_branch
Repository Root: https://code.metoffice.gov.uk/svn/um
Revision: 29416
Last Changed Author: rosalynhatcher
Last Changed Rev: 29416
Last Changed Date: 2016-10-21T10:30:48.533152Z
--------------------------------------------------------------------------------
Branch Create Author: rosalynhatcher
Branch Create Rev: 29406
Branch Create Date: 2016-10-21 10:26:42 +0100 (Fri, 21 Oct 2016)
--------------------------------------------------------------------------------
Branch Parent: https://code.metoffice.gov.uk/svn/um/main/trunk@24655
Merges Avail From Parent: 29409 29400 ....... 25507 25445 25365 25212 25077␣
↪24778
Merges Avail Into Parent: 29416
```

v. Testing that your branch works

Now that you have made a branch you can use it in the suite you were running earlier. Go back to the section where you added an existing branch to your suite and add your new branch as well.

Save and then **Run** your suite.

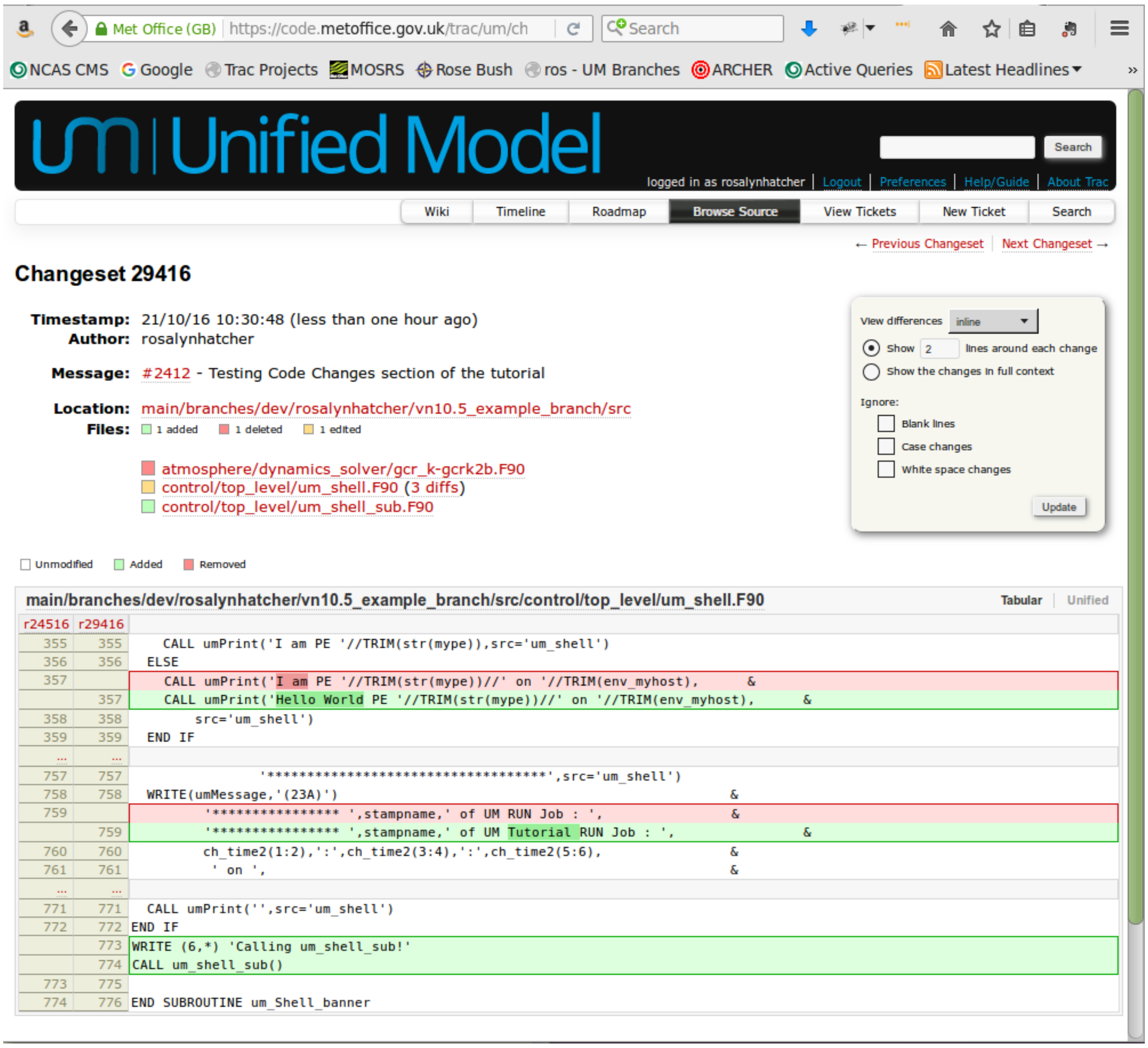
If you have followed the tutorial scenario so far you should find that your suite fails during the **fcm extract** of code. In the `job.err` file for the *fcm_make* task you will see an error message like this:

```
[FAIL] um/src/control/top_level/um_shell.F90: merge results in conflict
[FAIL]     merge output: /home/ros/cylc-run/u-ag954/share/fcm_make/
.fcm-make/extract/merge/um/src/control/top_level/um_shell.F90.diff
[FAIL]     source from location  0: svn://puma/um.xm_svn/main/trunk/src/
control/top_level/um_shell.F90@24655
[FAIL]     source from location  1: svn://puma/um.xm_svn/main/branches/dev/
rosalynhatcher/vn10.5_um_shell1/src/control/top_level/um_shell.F90@29416
[FAIL] !!! source from location  2: svn://puma/um.xm_svn/main/branches/dev/
rosalynhatcher/vn10.5_example_branch/src/control/top_level/um_shell.F90@29416
```

This is because the sample branch and your branch contain modifications to the same line in file `um_shell.F90` and so conflict. Errors like this can be quite common if you are working with others on the same section of code. The default behaviour of FCM in this situation is to fail and force you to resolve the conflict. For the purposes of this exercise we will simply remove the 'um_shell1' branch from the suite - we've decided we only want the changes we've put in our branch. In practice you will need to go through the process of resolving a conflict which can be quite complex. There is a tutorial dedicated to conflict resolution should you wish to know more and is a good reference should you encounter conflicts in your development work.

Viewing your changes in Trac

Making a change to your branch results in a **changeset** which is basically a record of the changes. One way of viewing the changeset you have just created is to click on **Timeline** in Trac. The Timeline view is a sequential record of all events in the repository. You should see changesets for your original commit to your branch and the subsequent commit after resolving the conflicts near the top. The changesets are numbered corresponding to the revision of your branch which would have been displayed in the GUI when you did a commit or branch info. To see all the details click on the line *'Changeset[xxx]... '* relating to your changeset. Alternatively, if you enter the number of the changeset "[**xxx**]" into the search box at the top right, it will take you directly to the numbered changeset. Your changeset should look something like this:



Changeset 29416

Timestamp: 21/10/16 10:30:48 (less than one hour ago)
Author: rosalynhatcher

Message: #2412 - Testing Code Changes section of the tutorial

Location: main/branches/dev/rosalynhatcher/vn10.5_example_branch/src
Files: 🟩 1 added 🟥 1 deleted 🟨 1 edited

🟥 atmosphere/dynamics_solver/gcr_k-gcrk2b.F90
🟨 control/top_level/um_shell.F90 (3 diffs)
🟩 control/top_level/um_shell_sub.F90

View differences: inline
(●) Show 2 lines around each change
( ) Show the changes in full context
Ignore:
☐ Blank lines
☐ Case changes
☐ White space changes
Update

☐ Unmodified 🟩 Added 🟥 Removed

main/branches/dev/rosalynhatcher/vn10.5_example_branch/src/control/top_level/um_shell.F90 — Tabular | Unified

```
r24516 r29416
 355    355      CALL umPrint('I am PE '//TRIM(str(mype)),src='um_shell')
 356    356    ELSE
 357             CALL umPrint('I am PE '//TRIM(str(mype))//' on '//TRIM(env_myhost),      &
        357      CALL umPrint('Hello World PE '//TRIM(str(mype))//' on '//TRIM(env_myhost),      &
 358    358          src='um_shell')
 359    359    END IF
 ...    ...
 757    757                   '***********************************',src='um_shell')
 758    758    WRITE(umMessage,'(23A)')                                                   &
 759                 '**************** ',stampname,' of UM RUN Job : ',                  &
        759          '**************** ',stampname,' of UM Tutorial RUN Job : ',         &
 760    760          ch_time2(1:2),':',ch_time2(3:4),':',ch_time2(5:6),                   &
 761    761           ' on ',                                                              &
 ...    ...
 771    771    CALL umPrint('',src='um_shell')
 772    772  END IF
        773  WRITE (6,*) 'Calling um_shell_sub!'
        774  CALL um_shell_sub()
 773    775
 774    776  END SUBROUTINE um_Shell_banner
```

4.5 Documenting your change

Go back to the Trac ticket you created for your code change and add some documentation as follows:

- Add a link to your branch:
  Development branch: `[source:main/branches/dev/<username>/<branch_name>]`
- A description of what code has changed
- Test results (i.e. Did your suite run? Were there any clashes to resolve?)
- Any other information you want to add
- As we have finished the change for this tutorial example we will resolve the ticket as **fixed** by clicking **Modify Ticket** and selecting *"resolve & assign to <username> as fixed"*.

Preview and **Submit** your ticket to save the changes. Check the link you added works.

4.6 Tidying Up

If your development is destined for the UM trunk, then once you have finished your code changes and it has been tested and reviewed, your branch will be committed to the project shared package branch by the project owner. Once this has been done and there are no problems, your branch is essentially redundant. If no other users are using this branch in their suites it can be deleted.

For the purposes of this tutorial, you can now proceed to delete your branch. When you delete a branch, it becomes invisible from the HEAD revision, but will continue to exist in the repository should you want to refer to it in the future.

List branches owned by you

If you forget what your branch is called and/or what other branches you have created, you can get a listing of all the branches you have created in a project. To do this use the following command:

```
fcm branch-list URL
```

Where URL is the name of repository you want to search. In this case it would be fcm:um.x

Delete a branch

Make sure you are in the relevant working copy directory and type:

```
fcm branch-delete
```

You will be prompted to edit the commit message file. Again a standard template is automatically supplied for the commit. Add your commit message, save your changes and exit the editor.

Answer **y** when you are prompted to go ahead and delete this branch.

Your working copy is now pointing to a branch that no longer exists at the HEAD revision of the repository. It is possible to keep this working copy, create a new branch and switch your working copy to point to the new branch. Otherwise, you can remove your working copy by issuing a careful `rm -rf` command.

4.7 Version Control of Suites

Just like the model code, your UM suites are also under version control in a subversion repository, usually *roses-u* which is on the MOSRS. Once you have a working copy of your suite under `~/roses` you can use FCM commands in the same way as for your source code branches; i.e. commit changes, diff changes, etc.

- Look in the roses-u repository via MOSRS Trac (https://code.metoffice.gov.uk/trac/roses-u) and find the suite you created in the previous section. (Hint: Go to *"Browse Source"* then drill down to find your suite. e.g. u-ag263 would be under a/g/2/6/3). When was the suite last modified?
- Go to your suite working directory and type **fcm status** to see the changes you have made since you copied the suite.
- Run **fcm commit** to commit your changes to the repository.
- Look again in the MOSRS roses-u Trac and see that your commit has now appeared in the repository. What is the suite's last modified time now?
- Use Trac to view the changes you have made to the suite. (Hint: click on the number in the revision column, and then on the *View changes* button to show a diff of your changes)

SECTION

FIVE

SOLVING COMMON UM PROBLEMS

This section exposes you to more typical UM errors and hints at how to find and fix those errors.

You may encounter other errors, often as a result of mistyping, for which solution hints are not provided.

5.1 Copy and set up N96 GA7.0 AMIP example suite

Find and make a copy of suite **u-cc654**.

Firstly make the changes required to run the suite. That is the account code ('n02-training'), your ARCHER user name and the queue to run in. Hint: Look in the *suite.conf* section. You will see that this suite has the queue reservations listed as Wednesday, Thursday, Friday; select the appropriate day.

- Did you manage to find where to set your ARCHER username?

This suite is set up slightly differently to the one used in the previous sections; suites do vary on how they are set up but you will soon learn where to look for things. This suite is set up so that specifying your username on the remote HPC is optional. If your PUMA username is the same as your username on ARCHER, or if the remote username is set in your `~/.ssh/config` file Cylc will be able to submit your suite without having to explicitly set your username in the suite. However, on this course, we are using training accounts on ARCHER so you will need to set the username.

- Click *View -> View Latent Variables*. You should see HPC_USER appear in the panel greyed out.
- Click the **+** sign next to it and select *Add to configuration*
- Enter your ARCHER training username (e.g. 'ncastr01')

5.2 Errors resolved in the code extraction

Save the suite and then **Run** it either from the GUI or the command line.

The suite should fail in the *fcm_make_um* task. This is the task that extracts all the required code from the repository including any branches. The failure will be indicated in the *gcylc* GUI with a red square and the state *failed*.

- What is the error?

Hint: Examine the *job.err* and *job.out* to find the cause of the problem. You can view these files through Rose Bush, as we have done previously, however you can also view them quickly and easily directly from the Cylc GUI. **Right-click** on the failed *fcm_make_um* task and select *View -> job stderr*

This indicates that the branch cannot be found due to an incorrect branch name. You will need to look at the UM code repository through Trac either on MOSRS (https://code.metoffice.gov.uk/trac/um/browser) or the PUMA mirror (https://puma.nerc.ac.uk/trac/um.xm/browser with username: guest1 and password: tra1n1ng or use your own) to determine the correct name.

Fix the error, **Save** the suite.

Now we will stop the suite and then re-run it. In the Cylc GUI click on *Control* -> *Stop Suite* and then select **Stop now** and then click on **OK**. **Run** the suite again.

The suite will fail in *fcm_make_um* again.

- What is the error?

Hint: Again look in the *job.err* file. This kind of error results when changes made in two or more branches affect the same bit of code and which the FCM system cannot understand how to resolve.

- Which file does the problem occur in?

In practice, you will need to fix the problem with the code conflict as you did in the FCM tutorial section. To proceed in this case, navigate to *fcm_make_um* -> *sources* and remove the branch called `vn11.7_training_merge_error` by clicking on it and then clicking the **-** sign.

Save the suite.

Last time we stopped the suite and then re-ran it, however, it is possible to reload the suite definition and then re-trigger the failed task without first stopping the running suite. To do this change to the suite directory:

```
puma$ cd ~/roses/<suitename>
```

We then reload the suite definition by running the following Rose command:

```
puma$ rose suite-run --reload
```

Wait for this command to complete before continuing. Finally in the Cylc GUI *right-click* on the failed task and select **Trigger (run now)**. The *fcm_make_um* task will then submit again.

- Is there an error in *fcm_make_um* this time?

If you look in the *job.err* file now it should be empty and the *job.out* file indicates SUCCESS.

5.3 Errors resolved in the compile and run

- Has the *fcm_make2_um* (compilation) task completed successfully?
- You should have a failure. Open the *job.err* file - what does it indicate?
- Which routine has an error?
- What is the error?
- What line of the Fortran file does it occur on?

In practice, you would need to fix the error in your branch on PUMA and then restart the suite. In this case, navigate to *fcm_make_um* -> *sources* and remove the branch `vn11.7_training_compile_error`. **Save** the suite, **Shutdown/Stop** the failed run and then **Run** it again. Notice we chose to shutdown the failed suite this time rather than do a reload. In this scenario we need to redo the code extraction (*fcm_make_um*) step so doing a reload would be slightly more complex; you would need to *Reload* and then *Re-trigger* both the *fcm_make_um* and the *fcm_make2_um* tasks. With experience you get to know when it's better to do a *Reload* and when to *Shutdown* a suite.

Note again that the task submitted successfully.

- Did the *fcm_make2_um* task succeed this time?
- What about the reconfiguration task?
- What is the error?
- Does the start dump exist?
- What is the name of the correct start dump? Hint: look in the directory where it thinks the start file should be - is there a candidate in there?

Point your suite to the correct start dump. Fixing this problem isn't quite as easy as it sounds. A search for **ainitial** in the Rose edit GUI will take you to the *General reconfiguration options* panel.

- Can you see the problem?

The initial dump location is set with an environment variable: AINITIAL. Suites can be and are set up differently and there will be times when you need to edit the cylc suite definition files directly.

In your suite directory on PUMA (`~/roses/<suitename>`) use `grep -R` to search for where the variable *AINITIAL* is set (If you are unfamiliar with using *grep* please ask for help). Edit AINITIAL in the appropriate `.rc` file to point to the correct initial dump file. (Hint: This suite is set up to run on multiple platforms, make sure you edit the file appropriate to ARCHER.) You may notice that AINITIAL is set 3 times; a different file is required depending on the resolution the model is being run at. This suite is running at N96 resolution.

Reload the suite definition and then **Re-trigger** the reconfiguration task. The reconfiguration should succeed this time.

- Has the model run successfully?

This time the model should have failed with an error.

- What is the error message?

Hint: Try searching for "ERROR" - you will soon learn common phrases to help track down problems.

Note: If you use the search `job.err` box at the bottom of the gcylc viewer, when you select *"Find Next"* you will see a message indicating the live feed will be disconnected. Click *Close*.

- At what timestep did the error occur?
- Which PE Ranks signalled the Abort? Make a note of which ones

Change to the `pe_output` directory for the atmos_main task. This is under `~/cylc-run/<suite-id>/work/<cycle>/atmos_main/pe_output` and contains the output from each PE.

Open the file called `<suite-id>.fort6.pe<pe noted above>`. Sometimes extra information about the error can be found in the individual PE output files.

The error message indicates that NaNs (NaN stands for Not a Number and is a numeric data type representing an undefined or unrepresentable value) have occurred in the routine EG_BICGSTAB. This basically means something in the model has become unstable and "blown up". In this case the failure results from an incorrect value for the solar constant *'sc'*. You could try to find what setting similar models use (with the MOSRS repository you have access to all model setups) or looking at the help within `rose edit` may point you in the right direction. Go to *um -> namelist -> UM Science Settings -> Planet Constants* and set it to the suggested value. **Save**, **Reload** and **Re-trigger**.

The model should fail with the same error. So what's gone wrong here? We've changed the value of the solar constant to a valid value so why didn't it work? The first thing to check is that the new value has indeed been passed to the model. We do this by checking the variable in the namelists which are written by the Rose system. On ARCHER navigate to the work directory for the *atmos_main* task (ie. `~/cylc-run/<suite-id>/work/<cycle>/atmos_main`). In here you will see several files with uppercase names (e.g. ATMOSCNTL, SHARED), these contain the Fortran namelists which are read into the model. Have a look inside one of them to see the structure. Now search (use *grep*) in these files for the solar constant variable *sc*. Hint: search for the string "*sc*=".

- What value does it have? Is this what you changed it to in the Rose edit GUI?

So why was the change not picked up? Go back to view the setting in the Rose GUI. By the side of the variable *sc* there is a little icon of a hand on paper, this indicates that there is an *"optional configuration override"* for this variable.

Optional configuration overrides add to or overwrite the default configuration. They are useful to make it easier to switch between different configurations of the model. For example switching between different resolutions.

Click on the icon and the list of overrides appears. You will see that the variable is set to 120000.0 in the *training* override file and it is this value that is being used in the model. Unfortunately optional configuration override files cannot be changed through the GUI so we will need to edit the Rose file directly. Override files for the *um* app live in the directory `~/roses/<suite-id>/app/um/opt`. Open the file `rose-app-training.conf` and edit the value for `sc`. **Save**, **Reload** and **Re-trigger** the suite.

Check the `sc` variable in the namelist file again to confirm that it does now have the correct value. This time the model should run successfully. Check the output to confirm that there are no errors. Check that the model converged at all time steps.

FURTHER EXERCISES (1)

Now that we have built the suite, there is no need to rebuild it each time you run it. Switch off compilation of the UM and reconfiguration. Hint: See the *"suite conf"* section.

6.1 Change the model output logging behaviour

Navigate to *um -> namelist -> Top Level Model Control -> Run Control and Time Settings*. Set **ltimer** to *true*. Timer diagnostics outputs timing information and can be very useful in diagnosing performance problems.

Save and **Run** the suite.

Check the *job.out* file.

- Which routine took the most time?
- How many times was *Atm_Step* called?
- How many time steps did the model run for?
- Which PE was the slowest to run AP2 Boundary Layer? Which was the fastest?

Switch on "IO timing" Hint: look in *"IO System Settings"*.

Re-run the model and search for "Total IO Timings" in the *job.out* file.

6.2 Change the processor decomposition

Navigate to *suite conf -> Domain Decomposition -> Atmosphere*.

- What is the current processor decomposition?
- Why is this not a good way to run the model?

Hint: The base ARCHER charging unit is a node irrespective of how many cores on the node are being used. ARCHER has 24 cores per node, and for the UM each MPI task and OpenMP thread is mapped to a separate core. So in this case we are running 8x10 (x 2 OMP threads) for a total of 160 cores, but are charged for 7 nodes (168 cores).

Try reducing the processor decomposition to 4(EW) x 3(NS). Run the model.

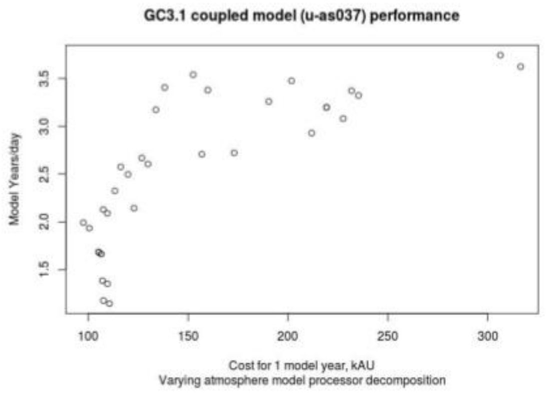
- What is the error?

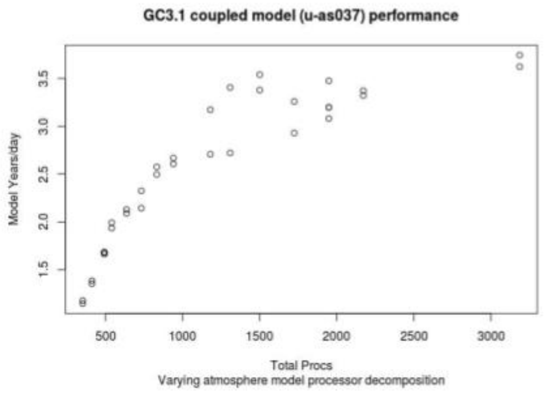
Using too few cores has resulted in the model failing due to insufficient memory, indicated by the message *"OOM killer terminated this process"*. In this case it is easy to diagnose the problem, but sometimes it can be difficult to diagnose, so it's worth trying to increase the number of processors if you suspect memory resource issues.

Try experimenting with different processor decompositions (E.g. 8x6, 12x12, etc)

- How do the timings compare to when you ran on 7 nodes?

You can come up with a performance vs processor count curve in this way which might be valuable if you are planning an experiment - it's also worth adding in the AU cost calculation when doing this. An example of this can be seen below:





Note: Running "under populated", i.e. with fewer than the total cores per node, gives access to more memory per parallel task.

Change the processor decomposition to run fully populated on 8 nodes with 2 OpenMP threads.

6.3 STASH

i. Exploring STASH

Navigate to *um* -> *namelist* -> *Model Input and Output* -> *STASH Requests and Profiles*. Look at the time profiles called `TALLTS` and `T1H`.

- What are they doing?

(`TALLTS` says output on every timestep, `T1H` says output hourly)

Look also at some of the other time, domain and usage profiles. The domain profiles determine spatial output and the usage profiles effectively specify a Fortran LUN (Logical Unit Number) on which the associated data is written.

Click on *STASH Requests*. Now change the time profile for all stash output whose `Usage` profile is UPC and `Time` profile is T1H - to do this, click on each diagnostic you wish to change and then click the time profile, a drop-down list should appear containing all the available time profiles. Select `TALLTS`. You can sort the STASH table to make it more convenient to make these changes. Click on the `use_name` column header to sort by usage profile.

ii. STASH validation macro

Several Rose macros have been provided to help verify STASH setup. When you change STASH it is always recommended to run at least the validate macro. The *stash_testmask.STASHTstmskValidate* macro ensures that the STASH output requsted is valid given the science configuration of the app. To put this to the test run the STASH validation macro by selecting *stash_testmask.STASHTstmskValidate* from the list of available macros at the top of the STASH requests panel or alternatively it can be accessed from the *"Metadata -> um"* menu.

You should see several errors reported - it appears we have asked for diagnostics which are not available. This won't cause the model to fail, however, you could find these diagnostics in the list and switch them off by unchecking the "incl?" column, if you'd like to stop seeing this message.

Save and **Re-run** the suite.

The model should fail with an error message similar to the following:

STWORK: Number of fields exceeds reserved headers for unit 14

This means that the number of output fields exceeds the limit set for a particular stream (the default is 4096 fields); in this case the stream attached to unit 14. To find out what stream unit 14 is take a look in the `job.out` file and search for "unit 14". You should see that the file opened on unit 14 is <suite-id>a.pc19880901, so this is the **pc** stream. Back in `rose edit` for this suite look at the STASH usage profile for **upc**.

- What is the file ID of the failing output stream?

Now navigate to the window for this stream under *Model Input and Output -> Model Output Streams*. This defines the output stream. You should see confirmation of the base output file name to be `*.pc*`. Changing the reinitialisation frequency by modifying **reinit_step** and/or **reinit_unit** is the best way to fix this header problem. This tells the model to create new output files at a specified frequency, so individual files don't get massively large.

Note: If the model is only exceeding the numer of reserved headers by a small amount it is also possible to just increased the **reserved_headers** size. Overriding the size by a large amount and thus having large numbers of fieldsfile headers can be very inefficient for both runtime and memory. Therefore the recommended way is to change the periodic reinitialisation of the fieldsfiles.

Modify the reinitialisation frequency (you will need to experiment with the numbers) and run the model again. Take a look at the model output files. You should see that you have multiple `*.pc19980901_*` files.

iii. Adding a new STASH request

Let's now try adding a new STASH request to the UM app.

Click the "New" button in the STASH Requests section. A window will appear in order for you to browse all available STASHmaster entries.

By default STASHmaster entries are grouped together by Section code. It is possible to group items by any of the STASHmaster codes using the Group drop down list. The View button contains options to display the STASHmaster entry values and/or the column titles with explanation text and to select which columns to show/hide.

Expand the *"Gravity wave drag"* section. Then change the view by selecting *View -> Show expanded value info*. Try out the other options in the *View* menu to see what effect they have.

Select a STASH item and click **Add** to add it to the list of STASH requests. In the STASH Requests panel click on the empty *dom_name*, *tim_name* and *use_name* fields of the new request and select appropriate profiles from the drop down lists. These lists are populated from the entries of the time, use and domain namelists.

Once you have added a new STASH request, you need to run a macro to generate an index for the namelist. To do so click on the **Macros** button, then select **stash_indices.TidyStashTransform**. A box will pop up listing the changes the editor is going to make, click **Apply**.

- Run the model. Did it work?

6.4 Change the dump frequency

Set the model run length to 2 hours. Hint: *suite conf -> Run Initialisation and Cycling*.

Note: Hours are represented in the ISO 8601 standard as *PT<num-hours>H* (e.g. PT1H represents 1 hour). Days are represented as P<num-days>D (e.g. P10D represents 10 days)

Reset the STASH output for stream UPC to hourly and the file reinitialisation frequency to 4 hourly.

Navigate to *um -> namelist -> Model Input and Output -> Dumping and Meaning*.

- What is the current dump frequency?

Set the dump frequency to 2 hours. **Run** the model.

- How much time was spent in DUMPCTL?

Set the dump frequency to 1 hour. **Run** the model.

- What happened to the time spent in DUMPCTL?

It is important to understand that writing out model dumps, particularly at higher resolutions, takes up a large amount of time and contributes to the cost. You should think about how frequently you need to output model dumps when setting up your simulations.

6.5 Reconfiguration

Try to find out how to run the reconfiguration only. Hint: Look in the *"suite conf"* section.

Try to find out where to request extra diagnostic messages for the reconfiguration output.

Run the reconfiguration only with extra diagnostic messages.

Look at the *job.out* file.

- Do you see a land-sea mask?

6.6 Setting up a suite to cycle

We mentioned in the presentations that the length of an integration will be limited by the time that a model is allowed to run on the HPC (see the ARCHER web pages for information about the time limits). Clearly this is no good for much of our work which may need to run on the machine for several months. Cylc and the UM allow for long integrations to be split up into multiple shorter jobs - this is called *cycling*.

Let's run the model for 3 hours with 1 hour cycling:

- Set the *"Total run length"* to 3 hours.
- Set the *"Cycling frequency"* to 1 hour.
- Set the *"Wallclock time"* to 10 minutes.
- Ensure that the model dump frequency is hourly, in this case.

Save and **Run** the suite.

Note: The cycling frequency must be a multiple of the dump frequency.

The model will submit the first cycle and once that has succeeded you will see the following 2 cycles submitted and run.

Note: It is always wise, particularly when you plan to run a long integration, that you only run the first cycle initially so that you can check that the model is doing what you expect before committing to a longer simulation. It also enables you to determine how long it takes your model to run and thus be able to calculate an appropriate cycling frequency for your simulation.

6.7 Restarting a suite

Let's now extend this run out to 6 hours. Change the *"Total run length"* to 6 hours and **Save** the suite.

Having already run the first 3 hours we just want the suite to pick up where it left off and run the remaining 3 hours. To do this we *restart* the suite, by typing:

```
puma$ rose suite-run --restart
```

The cylc GUI will pop up and you should see the run resuming from where it left off (i.e. from cycle point 19880901T0300Z).

SECTION

SEVEN

FURTHER EXERCISES (2)

The exercises in this section are all optional. We suggest you pick and choose the exercises that you feel are most relevant to the work you are/will be doing.

Note: Use your copy of suite u-ba799 for these exercises unless otherwise specified.

7.1 Post-Processing (archive and transfer of model data)

When your model runs it outputs data onto the Archer `/work` disk (`/projects` on Monsoon/NEXCS). If you are running a long integration and/or at high resolution data will mount up very quickly and you will need to move it either to the ARCHER RDF or JASMIN. The post-processing app (postproc) is used within cycling suites to automatically archive model data and can be optionally configured to transfer the data from the RDF to JASMIN data facility. The app archives and deletes model output files, not only for the UM, but also NEMO and CICE in coupled configurations.

Let's try configuring your suite to archive to the RDF:

- Switch on post-processsing in window *suite conf -> Tasks*

The post-processing is configured under the **postproc** section:

- Select the "Archer archiving system" in window *Post Processing - common settings*.

A couple of new entries will have appeared in the index panel, *Archer Archiving* and *JASMIN Transfer*, identified with the blue dots.

You now need to specify where you want your archived data to be copied to:

- In the *Archer Archiving* panel set `archive_root_dir` to be **/nerc/n02/n02/<userid>**. The *archive_name* (suite id) will be automatically appended to this.

You will need to run the model for at least 1 day as archiving doesn't work for periods of less than 1 day. Change the *"run length"* and *"cycling frequency"* to be 1 day. This should complete in about 5 minutes so set the *"wallclock time"* to be 10 minutes.

Run the suite.

Once the run has completed go to the archive directory for this cycle (e.g. `/nerc/n02/n02/<userid>/<suiteid>/19880901T0000Z`) and you should see several files have been copied over (e.g `ba902a.pc19880901_00.pp`). Data is only archived when it is no longer required by the model for restarting or for calculating means (seasonal, annual, etc). This run is reinitialising the `pc` data stream every 4 hours and you should see that it has only archived data files up to 16:00hrs (ba902a.pc19880901_16.pp). The last file containing data for hours 20-24 is still required by the model. Equally seasonal mean files would not be archived until the end of the year, after the annual mean has been created.

Note: The post-processing app can also be configured to transfer the archived data over to JASMIN. Details on how to do this are available on the CMS website: http://cms.ncas.ac.uk/wiki/Docs/PostProcessingApp

7.2 Using IO Servers

Older versions of the UM did not have IO servers, which meant that all reading and writing of fields files went through a single processor (pe0). When the model is producing lots of data and is running on many processors, this method of IO is very inefficient and costly - when pe0 is writing data, all the other processors have to wait around doing nothing but still consuming AUs. Later UM versions, including UM 10.5, have IO servers which are processors dedicated to performing IO and which work asynchronously with processors doing the computation.

Here's just a taste of how to get this working in your suite.

Set the suite to run for 6 hours with an appropriate cycling frequency, then check that OpenMP is switched on (Hint: search for *openmp* in rose edit) as this is needed for the IO servers to work.

Navigate to *suite conf* -> *Domain Decomposition* -> *Atmosphere* and check the number of OpenMP threads is set to 2. Set the number of *"IO Server Processes"* to 8.

Save and then **Run** the suite.

You will see lots of IO server log files in `~/cylc-run/<suitename>/work/<cycle>/atmos_main` which can be ignored for the most part.

Try repeating the "Change dump frequency" experiment with the IO servers switched on - you should see much faster performance.

7.3 Writing NetCDF output from the UM

Until UM vn10.9, only fields-file output was available from the UM - bespoke NetCDF output configurations did exist but not on the UM trunk. The suite used in most of these Section 7 exercises is vn11.0, hence supports both fields-file and NetCDF output data formats.

i. Enable NetCDF

Make sure that **IO Server Processes** is set to 0.

Navigate to *um* -> *namelist* -> *Model Input and Output* -> *NetCDF Output Options* and set `l_netcdf` to true. Several fields will appear which allow you to configure various NetCDF options. For this exercise, leave them at their chosen values.

ii. Set NetCDF Output Streams

Expand the *NetCDF Output Streams* section. A single stream - **nc0** - already exists; select it to display its content. As a useful comaprison, expand the *Model Output Streams* section and with the middle mouse button select **pp0**. Observe that the only significant differences between **pp0** and **nc0** are the values of `file_id` and `filename_base`. Data compression options for **nc0** are revealed if `l_compress` is set to true. NetCDF deflation is a computationally expensive process best handled asynchronously to computation and as yet not fully implemented through the UM IO Server scheme (but under active development.) For many low- to medium-resolution models and, depending precisely on output profiles, high-resolution models also, use of UM-NetCDF without IO servers still provides significant benefits over fields-file output since using it avoids the need for subsequent file format conversion.

Right-click on **nc0** and select *Clone this section*. Edit the settings of the newly cloned section appropriately to make the new stream similar to **pp1** (ie. edit `filename_base` and all the reinitialisation variables). It is sensible to change the name of the new stream from “1” to something more meaningful, nc1 for example (right click on “1”, select *Rename a section*, and change . . . nc(1) to . . . nc (nc1)).

iii. Direct output to the nc streams

Expand *STASH Requests and Profiles*, then expand *Usage Profiles*. Assign nc streams to usage profiles - in this suite, UPA and UPB are assigned to **pp0** and **pp1** respectively (where can you see this?). Edit these Usage profiles to refer to **nc0** and **nc1** respectively. Run the STASH Macros (if you need a reminder see Section 6), save the changes, and run the suite. Check that the NetCDF output is what you expected.

Try adding more nc streams to mimic the pp stream behaviour.

7.4 Running the coupled model

The coupled model consists of the UM Atmosphere model coupled to the NEMO ocean and CICE sea ice models. The coupled configuration used for this exercise is N96 resolution for the atmosphere and a 1 degree ocean - you will see this written N96 ORCA1.

i. Checkout and run the suite

Checkout and open the suite **u-ak943**. The first difference you should see is in the naming of the apps; there is a separate build app for the um and ocean, called *fcm_make_um* and *fcm_make_ocean* respectively. The model configuration is under *coupled* rather than *um*.

Make the usual changes required to run the suite (i.e. set username, account code, queue)

Check that the suite is set to build both the UM and ocean, as well as run the reconfiguration and model.

Run the suite.

ii. Exploring the suite

Whilst the suite is compiling and running which will take around 45 minutes, take some time to look around the suite.

- How many nodes is the atmosphere running on?
- How many nodes is the ocean running on?

Changing the processor decomposition for the ocean is not as simple as just changing the EW/NS processes. You also need to:

1. Recalculate the CICE number of columns per block EW and rows per block NS. (Normally the model is set up so that NEMO and CICE use the same decomposition). Looking at the current settings we calculate as follows:

   Num of cols per block EW = Num of cols EW / Num of processes EW (E.g. 360 / 9 = 40)

   Num of rows per block NS = Num of rows NS / Num of processes NS (E.g. 330 / 8 = 42)

2. Recompile the ocean executable. Note the executable comprises both the ocean (NEMO) and sea-ice (CICE) code.

Now looked at the `coupled` settings.

- Can you see where the NEMO model settings appear?

Look under *Run settings (namrun)*. The variables `nn_stock` and `nn_write` control the frequency of output files.

- How often are NEMO restart files written? (Hint the NEMO timestep length is set as variable `rn_rdt`).

Now browse the CICE settings.

- Can you find what the CICE restart frequency is set to?

NEMO and CICE are developed separately from the UM, and you should have seen that they work in very different ways. See the websites for documentation:

- http://oceans11.lanl.gov/trac/CICE
- http://www.nemo-ocean.eu/

iii. Output files

Log files

NEMO logging information is written to:

```
~/cylc-run/<suitename>/work/<cycle>/coupled/ocean.output
```

CICE logging information is written to:

```
~/cylc-run/<suitename>/work/<cycle>/coupled/ice_diag.d
```

If the model fails some error messages may also be written to the file `~/cylc-run/<suitename>/work/<cycle>/coupled/debug.root.01` or `debug.root.02`

When something goes wrong with the coupled model it can be tricky to work out what has gone wrong. NEMO errors may not appear at the end of the file but will be flagged with the string `E R R O R`.

Restart files

Restart files go to the subdirectories `NEMOhist` and `CICEhist` in the standard data directory `~/cylc-run/<suitename>/share/data/History_Data`.

Diagnostic files

Diagnostic files are left in the `~/cylc-run/<suitename>/work/<cycle>/coupled/` directory.

CICE files start with `<suitename>i`. Once your suite has run you should see the following CICE file:

```
archer$ ls ak943i*
ak943i.10d.1978-09-10.nc
```

NEMO diagnostic files are named `<suitename>o*grid_[TUVW]*`. To see what files are produced, run:

```
archer$ ls ak943o*grid*
```

In this case each processor writes to a separate file. To concatenate these into a global file use the `rebuild_nemo` tool, e.g.:

```
archer$ rebuild_nemo ak943o_10d_19780901_19780910_grid_W_19780901-19780910 72
```

Higher resolution NEMO suites may use the XIOS IO server. In this case, global files may be written directly, or each server process may write its own file.

Note: The coupled atmos-ocean model setup is complex so we recommend you find a suite already setup for your needs. If you find you do need to modify a coupled suite setup please contact NCAS-CMS for advice.

7.5 Running the Nesting Suite

The Nesting Suite drives a series of nested limited area models (LAM) from a global model. It allows the user to specify the domains and it then automatically creates the required ancillary files and lateral boundary condition files.

i. Checkout and run the suite

Checkout and open the suite **u-ba621**. There are a number of tasks for creating ancillary files (*ancil_** and *ants_**). The global model set up is in *glm_um* and the LAMs are in *um*. The task *um-createbc* creates the lateral boundary condition files.

Under *suite conf* -> *jinja2:suite.rc* are the main panels for controlling the Nesting Suite. Make the usual changes required to run the suite (i.e. set username, account code, queue). The training nesting suite has pre-built executables so you don't have to spend time building it. **Run** the suite.

This particular suite has a global model and one limited area model. It should complete in about 45 - 60 minutes.

ii. Exploring the Suite

The Driving Model set up panel allows the user to specify the resolution of the global model and the number of nested regions.

The *Nested Region 1* set up panel specifies the latitude and longitude of the centre of the first nested region. All the other limited area models have the same centre.

A useful way to get this information is to use Google Maps. Find the place you want as a centre and then press `control-left mouse` and a little window with the latitude and longitude appears.

- Can you find out where the first LAM is located? Hint: look at the orography file output during the ancillary creation.

The *resolution 1* set up panel specifies the grid and the run length.

The *Config 1* set up panel specifies the science configuration to be run. Each LAM can have multiple science configurations.

iii. Initial Data

The initial data for the global model is in `share/cycle/<cycle time>/glm/ics`

The initial data for the first LAM is in `share/cycle/<cycle time>/Regn1/resn_1/RA1M/ics`

The RA1M is the name you gave to the first science configuration.

The LBCs for the first LAM are in `share/cycle/<cycle time>/Regn1/resn_1/RA1M/lbcs`.

iv. The ancillary files

These are in `share/data/ancils/Regn1/resn_1`

v. The output files

The global model output is in `share/cycle/<cycle time>/glm/um`. This also contains contains the data for creating the LBC files (umglaa_cb*) for the first LAM.

Diagnostic files can be found under `work/<cycle time>` in an application directory. For example, the region1 forecast diagnostics is in `work/<cycle time>/Regn1_resn_1_RA1M_um_fcst_000`. This will include the pe_output files.

The output for the first LAM is in `share/cycle/<cycle time>/Regn1/resn_1/RA1M/um`.

vi. Further Information

This has been a very brief overview of the functionality of the Nesting Suite. The Nesting Suite is developed and maintained by Stuart Webster at the Met Office. He has a web page all about the Nesting Suite at https://code.metoffice.gov.uk/trac/rmed/wiki/suites/nesting. This includes a more detailed tutorial.

SECTION EIGHT

ROSE/CYLC EXERCISES

8.1 Differencing suites

Currently there is no Rose tool to difference two suites. Since a suite consists of text files it is simply a matter of making sure all the Rose configuration files are in the common format by running `rose config-dump` on each suite and then running `diff`.

We will difference your copy of the GA7.0 suite with the original one:

```
puma$ cd ~/roses
puma$ rosie checkout u-ba799
puma$ rose config-dump -C u-ba799
puma$ rose config-dump -C <your-suitename>
puma$ diff -r u-ba799 <your-suitename>
```

- Are the differences what you expected?

8.2 Graphing a suite

When developing suites, it can be useful to check what the run graph looks like after jinja evaluation, etc.

The GA7.0 suite that we have been working with is very simple so we shall graph a nesting suite which is more complex. To do this without running the suite:

```
puma$ rosie checkout u-ah076
puma$ cd ~/roses/u-ah076
puma$ rose suite-run -l --name=u-ah076 # install suite in local cylc db only
puma$ cylc graph u-ah076               # view graph in browser
```

A window containing the graph of the suite should appear.

8.3 Exploring the suite definition files

Change to the `~/roses/<suite-id>` directory for your copy of u-ag263.

Open the `suite.rc` file in your favourite editor.

Look at the `[scheduling]` section. This contains some Jinja2 variables (BUILD & RECON) which allow the user to select which tasks appear in the dependency graph. The dependency graph tells Cylc the order in which to run tasks. The `fcm_make` and `recon` tasks are only included if the `BUILD` and `RECON` variables are set to true. These variables are located in the `rose-suite.conf` and can be changed using the rose edit GUI or by directly editing the `rose-suite.conf` file. When you run a suite, a processed version of the `suite.rc` file, with all the Jinja2 code evaluated, is placed in your suite's `cylc-run` directory.

- Take a look at the `suite.rc.processed` file for your suite. Hint: go to directory `~/cylc-run/<suite-id>`.
- Change the values of BUILD and RECON and re-run your suite.
- Look at the new `suite.rc.processed` file. Can you see how the graph has changed?

Make sure that you leave the suite with BUILD=false before continuing.

As we saw earlier when changing the path to the start dump, some settings can't be changed through the rose edit GUI. Instead you have to edit the suite definition files directly.

- Can you find where the atmos processor decomposition is set for this suite?
- Change atmos processor decomposition to run on 2 nodes. Run the suite.
- What error message did you get? Hint: Look in the usual `job.out`/`job.err` or it may be in the `job-activity.log` file.

This error is caused by a mismatch in the number of nodes requested by the PBS job script header and the number of processors requested by the `aprun` command which launches the executable. (For further information on PBS and the aprun command on ARCHER see: http://www.archer.ac.uk/documentation/user-guide/batch.php).

In the `[[atmos]] [[[directives]]]` section change `-l select=1` to `-l select=2` to tell the PBS scheduler that you require 2 nodes.

- The suite should run this time. Did it run on 2 nodes as requested?
- How much walltime has been requested for the reconfiguration?

Now take a look at the `suite.rc` file for your other suite (the one copied from u-ba799). See how it differs. This one is set up to run on multiple platforms.

- Can you see the more complex dependency graph?
- Can you see where to change the reconfiguration walltime for this suite?

This has just given you a very brief look at the suite definitions files. More information can be found in the cylc documentation.

8.4 Suite and task event handling

Suites can be configured to send emails to alert you to any task or suite failures (or indeed when the suite finishes successfully). To send an email, you use the built-in setting `[[[events]]] mail events` to specify a list of events for which notifications should be sent. Here we will configure your copy of suite u-ba799 to send an email on task (submission) failure, retry and timeout.

Edit the `suite.rc` file to add the `[[[events]]]` section below:

```
[runtime]
    [[root]]
        ...
        [[[environment]]]
        ...
        [[[events]]]
            mail events = submission retry, retry, submission failed, failed,␣
↪submission timeout, timeout
            submission timeout = P1D
```

Configure cylc so it knows what your email address is. Edit the file `~/.cylc/global.rc` (create it if it doesn't exist) to add the following:

```
[task events]
    mail to = <enter-your-email-address>
```

To test this out we need to force the suite to fail. Change the account code to a non-existent one; e.g. 'n02-fail'

- Did you get an email when the suite failed?
- Look in the suite error files to find the error message?

Change the account code back to 'n02-training' before continuing.

Further information about event handlers can be found in the Cylc documentation: https://cylc.github.io/doc/built-sphinx-single/index.html#eventhandling

8.5 Starting a suite in "held" mode

This allows you to trigger the running of tasks manually.

To start a suite in held mode add `-- --hold` to the end of the `rose suite-run` command:

```
puma$ rose suite-run -- --hold
```

The first -- tells Rose that all subsequent options should be passed on to Cylc. This is why the hold option should be added to the end of the command, after any Rose options. Once the suite has started all tasks will be in a held state. It is then possible to select which tasks are run by right clicking on a task in the Cylc GUI and manually triggering it or resetting its state.

Try doing this as a way to run the reconfiguration only in one of your suites.

8.6 Discovering running suites and the multi-suite monitor GUI

Suites that are currently running can be detected with command line or GUI tools:

Submit 2 of your suites. It doesn't matter what tasks they are running for this exercise; compilation, recon or model run.

Now try running the command `cylc scan`. This lists your currently running suites. For example:

```
puma$ cylc scan
u-af140 ros@localhost:7770
u-ag761 ros@localhost:7776
```

There is also a multi-suite monitor GUI, which allows you to monitor the states of all suites you have running in one window. Try running the command:

```
puma$ cylc gscan &
```

Double clicking on a suite in *gscan* opens the *gcylc* window, which you will be very familiar with by now. For each suite open the *gcylc* window and stop the suite by going to *Control* -> *Stop Suite*, selecting **Stop after killing active tasks** and clicking **Ok**.

8.7 Adding a new app to a suite

A Rose application or "Rose app" is a Rose configuration for running an executable command, encapsulating details such as scripts, programs and settings.

To add a new app to a suite, we first create a directory to hold the app files. The main details are specified in a configuration file `rose-app.conf`. We may also specify some metadata to tell the general user what inputs to the task mean (this goes under a `meta/` sub-directory or we may reference some standard metadata held elsewhere). Any scripts or executables needed by the new app can be added into an app `bin/` directory. General scripts that aren't specific to the app should go in the *suite* `bin/` directory.

Remember to `fcm add` any new files that you add to the suite so they will be added to the repository when you next commit.

In order to actually run the app, we need to add a new "task" to the suite which involves editing the suite configuration file `suite.rc`. We need to specify 3 things:

1) how the new task relates to other tasks, specifically, which task will trigger it and which task will follow it;

2) what the task will run (i.e which app); and

3) how the task will run (i.e. which computer and the resources it will need).

In this example, we will add an app that prints `Hello World`, which will execute after the reconfiguration and before the main model. We will add the app to your copy of u-ba799.

i. Create the Rose application directory

Make sure the Rose edit GUI for your suite is closed. `cd` into the suite `app/` directory and create a new directory called `new_app`

```
puma$ cd ~/roses/<SUITEID>/app
puma$ mkdir new_app
```

ii. Create the Rose app configuration file

Change into the `new_app` directory and create a blank app configuration file called `rose-app.conf`:

```
puma$ touch rose-app.conf
```

Start the Rose editor (remember you need to be in the top level of the suite directory). You should now see the new application listed in the left hand panel. At this point it is an empty application and is not integrated into the task chain. Click on the little triangle to the left of *new_app* to expand its contents (you may need to select *View* -> *View Latent Pages* to see this). Everything is greyed out. Click on `command` to see the command page and then click the plus sign next to "command default" (you may need to select *View* -> *View Latent Variables* to see it) and select "add to configuration" to add a command to the application. Enter `echo "Hello World"` in the "command default" box. Save this and then have a look at the contents of the `rose-app.conf` file to see the effect.

iii. Add a new task to the suite definition

In order to execute the app, we need to add a new task to the suite workflow. This task executes our new application on a machine that we specify. In this instance we are adding the new task between the reconfiguration and the model run, and the task will be run on ARCHER in the serial queue.

To set this up, edit the `suite.rc` file. Under,

```
[scheduling]
   [[dependencies]]
```

find the line

```
graph = recon  => atmos_main
```

and change it to

```
graph = recon => hello => atmos_main
```

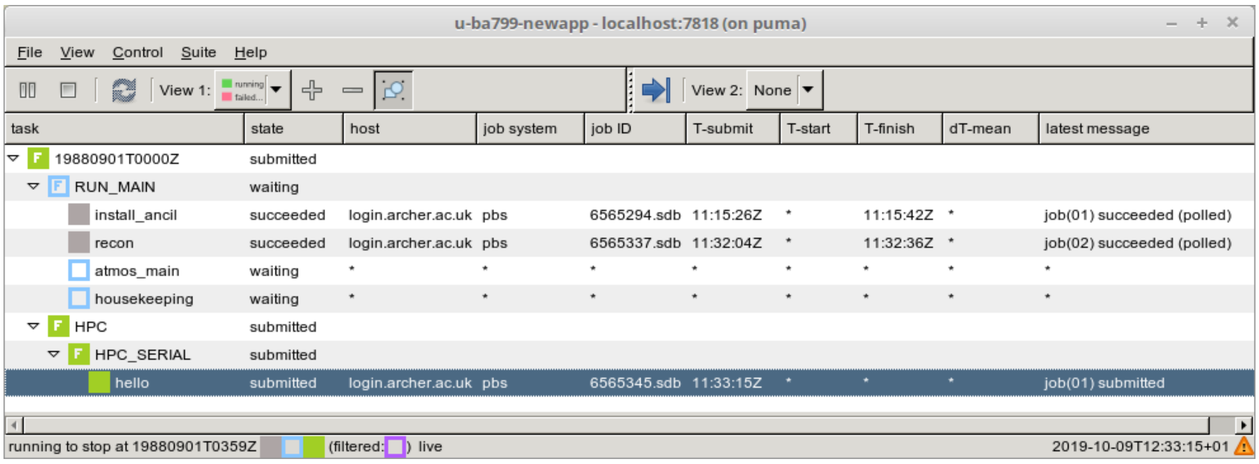
This puts the task `hello` in the right place in the task list.

The next step is to add a definition for the new task. To tell Rose to use one of the apps contained in the suite, we set the environment variable `ROSE_TASK_APP` in the task definition. General task definitions go in the `suite.rc` file and the definitions specific to ARCHER in the `site/archer.rc` file. The queuing system is specific to the host being run on, and there is already a definition for the ARCHER serial queue environment `[[HPC_SERIAL]]` that we can make use of. To run the new application on ARCHER in the serial queue and give it two minutes to complete, add the following lines to the `suite.rc` after the definition for `[[recon]]`:

```
[[hello]]
   inherit = HPC_SERIAL
   [[[environment]]]
      ROSE_TASK_APP = new_app
   [[[job]]]
      execution time limit = PT2M
```

iv. Running the new app

We are now ready to go. **Run** the suite. Look at the task graph: recon and atmos_main are there, but a new hierarchy of tasks has appeared.


u-ba799-newapp - localhost:7818 (on puma)
File View Control Suite Help
View 1:
View 2: None

task	state	host	job system	job ID	T-submit	T-start	T-finish	dT-mean	latest message
19880901T0000Z	submitted								
RUN_MAIN	waiting								
install_ancil	succeeded	login.archer.ac.uk	pbs	6565294.sdb	11:15:26Z	*	11:15:42Z	*	job(01) succeeded (polled)
recon	succeeded	login.archer.ac.uk	pbs	6565337.sdb	11:32:04Z	*	11:32:36Z	*	job(02) succeeded (polled)
atmos_main	waiting	*	*	*	*	*	*	*	*
housekeeping	waiting	*	*	*	*	*	*	*	*
HPC	submitted								
HPC_SERIAL	submitted								
hello	submitted	login.archer.ac.uk	pbs	6565345.sdb	11:33:15Z	*	*	*	job(01) submitted

running to stop at 19880901T0359Z (filtered:) live 2019-10-09T12:33:15+01

Notice that `atmos_main` no longer runs after the reconfiguration, but our new task `hello` does and when that has completed, `atmos_main` starts. The output from the `hello` task can be found in the cylc output directory: `log/job/19880901T0000Z/hello/NN/job.out`.

v. Extending the app to run a script

A more complex application might involve the execution of a script. To do this we would replace the contents of the "command default" box with the name of the script. Then place the script in the app `bin/` directory.

Now create a `bin/` directory under `new_app/` and `cd` into it. Create a file called `hello.sh` with the contents,

```
#!/bin/bash
echo "Hello, $1!"
```

We will allow the user to select from a variety of planets and say hello. Make it an executable script:

```
chmod +x hello.sh
```

Then we can say `./hello.sh Jupiter` to get it to print "Hello, Jupiter!".

Right click on the greyed out *new_app* -> *env* in the index panel and click "+ Add env". **Save**, then select *new_app* -> *env* to view the `env` page, right click on the blank page and select "Add blank variable". Two boxes appear: enter **PLANET** in the first and **Jupiter** in the second. This adds an environment variable called `PLANET` and sets it to "Jupiter".

Now change the command from echo "Hello, World" to hello.sh ${PLANET}.

vi. Testing and Running

The app can be tested in isolation by changing into the `new_app/` directory and executing,

```
rose app-run
```

This should produce output similar to:

```
ros@puma$ rose app-run
[INFO] export PATH=/home/ros/roses/u-ba799/app/new_app/bin:/home/fcm/rose-
↪2016.11.1/bin:/usr/local/python/bin:
...
[INFO] export PLANET=Jupiter
[INFO] command: hello.sh ${PLANET}
Hello, Jupiter!
```

and also a file `rose-app-run.conf`, which can be deleted.

Now **run** the suite.

vii. Rose Metadata

Metadata can be used to provide information about settings in Rose configurations. It is used for documenting settings, performing automatic checking and for formatting the rose edit GUI. Metadata can be used to ensure that configurations are valid before they are run.

Metadata for many standard applications, such as `um-atmos`, `fcm_make` are all stored centrally on PUMA in `~fcm/rose-meta`. Have a look at this directory.

For our example there are currently no restrictions on the variable `PLANET`. We will now add some metadata to help the user understand what the variable `PLANET` is and what values it is limited to.

Rose provides some tools to quickly guess at the metadata where there is none. Create a directory `meta/` under `new_app/` . Then execute the command,

```
rose metadata-gen
```

This creates a file `rose-meta.conf` in the `meta/` directory. It just says that there is an evironment variable called `PLANET`, but it does not know much about it. Edit this file and add the following lines after `[env=PLANET]`:

```
description=The name of the planet to say hello to.
values=Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune
help=Must be a planet bigger than Pluto - see https://en.wikipedia.org/wiki/
↪Solar_System
```

Now go back to the Rose GUI and select *Metadata -> Refresh Metadata*. Once the metadata has reloaded, go to the *new_app -> env* panel. The entry box for `PLANET` has changed into a drop down list. Pluto is not allowed, presumably because the code cannot handle tiny planets. Right click on the cog next to Planet and select `info` to see the description and allowed values.

viii. References

A fuller discussion of Rose metadata can be found at https://metomi.github.io/rose/doc/html/tutorial/rose/metadata.html.

Designing a new application may seem a daunting process, but there are numerous existing examples in suites that you can try to understand. For further details, see the Rose documentation at https://metomi.github.io/rose/doc/html/tutorial/rose/applications.html. There are a collection of built-in applications that you can use for building, testing, archiving and housekeeping - see https://metomi.github.io/rose/doc/html/api/rose-built-in-applications.html.

SECTION

NINE

POST PROCESSING

This is simply a very basic introduction to some of the more widely used useful tools for viewing, checking, and converting UM input and output data. The tools described below all run on the ARCHER login nodes, and you can run them there, but they also run on the ARCHER post-processors (see http://www.archer.ac.uk/documentation/user-guide/connecting.php#sec-2.1.2). Try logging on to one of the post-processors for these exercises: `ssh -X espp1`. The post processors can see /home and /work and /nerc.

9.1 xconv

i. View data

On ARCHER go to the output directory of the global job that you ran previously (the one copied from u-ba799). Run `xconv` on the file ending with, for example, `da19880901_04`. This file is an atmosphere start file - this type of file is used to restart the model from the time specified in the file header data.

In the same directory is a file whose name ends in `.astart`; run a second instance of `xconv` on this file. This is the file used by the model to start its run - created by the reconfiguration program in this case.

The `xconv` window lists the fields in the file, the dimensions of those fields (upper left panel), the coordinates of the grid underlying the data, the time(s) of the data (upper right panel), some information about the type of file (lower left panel), and general data about the field (lower right panel.)

Both files have the same fields. Double click on a field to reveal its coordinate data. Check the time for this field (select the "t" checkbox in the upper right panel).

Plot both sets of data - click the "Plot Data" button.

View the data - this shows numerical data values and their coordinates and can be helpful for finding spurious data values.

ii. Convert UM fields data to netCDF

Select a single-level field (one for which nz=1), choose "Output format" to be "Netcdf", enter an "Output file name", and select "Convert". Information relevant to the file conversion will appear in the lower left panel.

Use `xconv` to view the netcdf file just created.

9.2 uminfo

You can view the header information for the fields in a UM file by using the utility `uminfo` - redirect the output to a file or pipe it to `less`:

```
archer$ uminfo <one-of-your-fields-files> | less
```

The output from this command is best viewed in conjunction with the Unified Model Documentation Paper F3 which explains in depth the various header fields.

9.3 Mule

Mule consists of a Python API for reading and writing UM files and a set of UM utilities. This section introduces you to some of the most useful UM utilities. Full details of Mule can be found on the MOSRS: https://code.metoffice.gov.uk/doc/um/index.html

Before running the mule commands you will need to load the python environment on ARCHER by running:

```
archer$ module load anaconda
```

i. mule-pumf

This provides another way of seeing header information, but also gives some information about the fields themselves. Its intended use is to aid in quick inspections of files for diagnostic purposes.

Run `mule-pumf` on the start file - here's a couple of examples on one of Ros' files:

```
archer$ mule-pumf --print-columns 2 --headers-only \\
                      ba799.astart > ~/mule-pumf-header.out

archer$ mule-pumf --print-columns 2 ba799.astart > ~/mule-pumf.out
```

- Can you see what the difference is in the output of these 2 commands?

Take a look at the man page (`mule-pumf -h`) and experiment with some of the other options

ii. mule-summary

This utility is used to print out a summary of the lookup headers which describe the fields from a UM file. Its intended use is to aid in quick inspections of files for diagnostic purposes.

Run `mule-summary` on the start file again.

iii. mule-cumf

This utility is used to compare two UM files and report on any differences found in either the headers or field data. Its intended use is to test results from different UM runs against each other to investigate possible changes. Note, differences in header information can arise even when field data is identical. Try out the following:

- Run `mule-cumf` on the two start files referred to above (in the *"View data"* section). You may wish to direct the output to a file.
- Run the same command but with the `--summary` option. This, as the name suggests, prints a much shorter report of the differences.
- Run `mule-cumf` on a file and itself.

- View the help page with `mule-cumf -h` to find view all the available options.

9.4 um-convpp

We have mentioned in the presentations the PP file format - this is a sequential format (a fields file is random access) still much used in the community. PP data is stored as 32-bit, which provides a significant saving of space, but means that a conversion step is required from a fields file (64-bit). The utility to do this is called `um-convpp`. `um-convpp` converts directly from 64-bit files produced by the UM to 32-bit PP files. You must, however, make sure you are using a version 10.4 or greater - you can check that you are using the right one by typing `which um-convpp`.

Add the path to `um-convpp` to your environment - you can also add this to your `~/.profile` so it is available everytime you log in.

```
archer$ export PATH=$UMDIR/vn11.0/cce/utilities:$PATH
```

Run `um-convpp` on a fieldsfile (E.g ba799a.pc19880901_00)

```
archer$ cd /home/n02/n02/ros/cylc-run/u-ba799/share/data/History_Data
archer$ um-convpp ba799a.pc19880901_00 ba799a.pc19880901_00.pp

archer$ ls -l ba799a.pc19880901*
-rw-r--r-- 1 ros n02 64917504 Mar 15 11:56 ag761a.pc19880901_00
-rw-r--r-- 1 ros n02 48581456 Mar 21 10:19 ag761a.pc19880901_00.pp
```

Note the reduction in file size. Now use xconv to examine the contents of the PP file.

9.5 cfa

There is an increasing use of python in the community and we have, and continue to develop, python tools to do much of the data processing previously done using IDL or MATLAB and are working to extend that functionality. `cfa` is a python utility which offers a host of features - we'll use it to convert UM fields file or PP data to CF-compliant data in NetCDF format. You first need to set the environment to run `cfa`:

```
esPP001$ export PATH=/home/n02/n02/ajh/anaconda3/bin:$PATH
esPP001$ cfa -i -o ba799a.pc19880901_00.nc ba799a.pc19880901_00.pp
```

Try viewing the NetCDF file with xconv.

`cfa` can also view CF fields. It can be run on PP or NetCDF files, to provide a text representation of the CF fields contained in the input files. Try it on a PP file and its NetCDF equivalent, e.g.

```
archer$ cfa -vm ba799a.pc19880901_00.pp | less
Field: long_name:HEAVYSIDE FN ON P LEV/UV GRID (ncvar%UM_m01s30i301_vn1100)
---------------------------------------------------------------------------
Data            : long_name:HEAVYSIDE FN ON P LEV/UV GRID(time(5), air_
↪pressure(17), latitude(145), longitude(192))
Cell methods    : time: point
Axes            : time(5) = [1988-09-01T00:00:00Z, ..., 1988-09-01T03:59:59Z]␣
↪360_day
                : air_pressure(17) = [1000.0, ..., 10.0] hPa
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
```

(continues on next page)

(continued from previous page)

```
                 : longitude(192) = [0.0, ..., 358.125] degrees_east

Field: long_name:VORTICITY 850 (ncvar%UM_m01s30i455_vn1100)
------------------------------------------------------------
Data             : long_name:VORTICITY 850(time(5), latitude(145),␣
↪longitude(192))
Cell methods     : time: point
Axes             : air_pressure(1) = [-1.0] hPa
                 : time(5) = [1988-09-01T00:00:00Z, ..., 1988-09-01T03:59:59Z]␣
↪360_day
                 : latitude(145) = [-90.0, ..., 90.0] degrees_north
                 : longitude(192) = [0.0, ..., 358.125] degrees_east
```

9.6 CF-python CF-plot

Many tools exist for analysing data from NWP and climate models and there are many contributing factors for the proliferation of these analysis utilities, for example, the disparity of data formats used by the authors of the models, and/or the availability of the underlying sofware. There is a strong push towards developing and using python as the underlying language and CF-netCDF as the data format. CMS is home to tools in the CF-netCDF stable - here's an example of the use of these tools to perform some quite complex data manipulations. The user is insulated from virtually all of the details of the methods allowing them to concentrate on scientific analysis rather than programming intricacies.

- Set up the environment and start python.

```
archer$ export PATH=/home/n02/n02/ajh/anaconda3/bin:$PATH
archer$ python
>>> import cf
```

We'll be looking at CRU observed precipitation data

- Read in data files

```
>>> f = cf.read('~charles/UM_Training/cru/*.nc')[0]
```

- Inspect the file contents with different amounts of detail

```
>>> f
>>> print(f)
>>> f.dump()
```

Note that the three files in the cru directory are aggregated into one field.

- Average the field with respect to time

```
>>> f = f.collapse('T: mean')
>>> print(f)
```

Note that the time coordinate is now of length 1.

- Read in another field produced by a GCM, this has a different latitude/longitude grid to regrid the CRU data to

```
>>> g = cf.read('~charles/UM_Training/N96_DJF_precip_means.nc')[0]
>>> print(g)
```

- Regrid the field of observed data (f) to the grid of the model field (g)

```
>>> f = f.regrids(g, method='bilinear')
>>> print(f)
```

- Subspace the regridded field, f, to a European region

```
>>> f = f.subspace(X=cf.wi(-10, 40), Y=cf.wi(35, 70))
>>> print(f)
```

Note that the latitude and longitude coordinates are now shorter in length.

- Import the cfplot visualisation library

```
>>> import cfplot
```

- Make a default contour plot of the field, f

```
>>> cfplot.con(f)
```

- Write out the new field f to disk

```
>>> cf.write(f, 'cru_precip_european_mean_regridded.nc')
```

This has just given you a taster of CF-Python & CF-Plot, if you would like to try out some more exercises please take a look at http://ajheaps.github.io/cf-plot

SECTION

TEN

APPENDIX A: USEFUL INFORMATION

10.1 UM output

ARCHER job output directory:

The standard output and error files (job.out & job.err) for the compile, reconfiguration and run are written to the directory:

```
~/cylc-run/<suite-id>/log/job/<cycle>/<app>
```

ARCHER model output:

By default the UM will write all output (e.g. processor output and data files) to the directory it was launched from, which will be the task's **work** directory. However, all output paths can be configured in the GUI and in practice most UM tasks will send output to one or both of the suite's **work** or **share** directories:

```
~/cylc-run/<suitename>/work/1/atmos
~/cylc-run/<suitename>/share/data
```

10.2 ARCHER architecture

ARCHER has two kinds of processor which we commonly use - they have several names, but roughly speaking they are the service processors (several nodes worth) sometimes referred to as the front end, and the compute processors (many many nodes worth) sometimes referred to as the back end. We login to the front end and build the model on the front end. We run the model on the back end. You wouldn't generally have an interactive session on the back end and will submit jobs there through the batch scheduler (PBS).

The UM infrastructure recognises this architecture and will run tasks in the appropriate place.

If you are doing any post-processing or analysis you may wish to submit your own parallel or serial jobs. Intensive interactive tasks should be run on the post-processor nodes. For analysing data on the /nerc disk, use the RDF cluster.

Consult the ARCHER documentation for details.

10.3 ARCHER file systems

ARCHER, in common with some other HPC systems, such as MONSooN and Polaris, has (at least) two file systems which have different properties, different uses, different associated policies and different names. On ARCHER there are `/home` and `/work`. The `/home` file system is backed up regularly (only for disaster recovery), has relatively small volume, can efficiently handle many small files, and is where we recommend the UM code is saved and built. The `/home` system can not be accessed by jobs running on the compute processors.

The `/work` file system is optimized for fast parallel IO - it doesn't handle small files very efficiently. It is where your model will write to and read from.

10.4 ARCHER node reservations

In normal practice you will submit your jobs to the parallel queue on ARCHER; the job scheduler will then manage your job request along with all those from the thousands of other users. For this training course, we will be using processor Reservations, whereby we have exclusive access to a prearranged amount of ARCHER resource meaning that you will not need to wait in the general ARCHER queues. Reservations are specified by a reservation code - e.g. R4943949. As an ARCHER user you can make a reservation so that you have access to the machine at a time of your choosing - reservations incur a cost overhead (50%), so best used when you are sure you need them.

10.5 Useful Rose commands

`rose suite-run`

Run a suite.

`rose suite-run --new`

Clear out any existing cylc-run directories for this suite and then run it. Take care when using this option as it deletes all files from any previous runs of the suite.

`rose suite-run --no-log-archive`

Do not archive (tar-gzip) old log directories.

`rose suite-run --restart`

Restart the suite from where it finished running previously

`rose suite-run [--restart] -- --hold`

Hold (don't run tasks) immediately on running or restarting the suite

`rose suite-shutdown`

Shutdown (stop) a running suite.

`rose sgc`

Launch the Cylc GUI for a running suite.

`rose suite-scan`

Scan for any running suites. This is useful when you've shutdown the cylc GUIs and wish to quickly see what suites you still have running.

For more information on all these commands and more see the Rose and Cylc documentation or run `rose command --help` (E.g. `rose suite-run --help`) to view the man pages.

10.6 Problems shutting down suites

Types of shutdown

By default when you try to shutdown a suite, cylc will wait for any currently running tasks to finish before stopping, which may not be what you want to do. You can also tell cylc to kill any active processes or ignore running processes and force the suite to shutdown anyway. The latter is what you will need to do if the suite has got stuck:

```
rose suite-shutdown -- --now
```

To access these options in the cylc GUI, go to *"Control"* -> *"Stop Suite"*. See also `rose help suite-shutdown` for further details.

Forcing shutdown

Sometimes after trying to shutdown a suite, it will still appear to be running.

First make sure you have used the correct shutdown command and aren't waiting for any unfinished tasks (see above). It can take cylc a little while to shut down everything properly, so be patient and give it a few minutes.

If it still appears to be running (for example you get an error when you try to re-start the suite), you may have to do the following:

- Manually kill the active processes:

  Get a list of processes associated with the suite. For example, for suite u-ak194 you would run:

  ```
  puma u-ak193$ ps -flu annette  | grep u-ak194
  0 S annette   2735  5230  ... grep u-ak194
  1 S annette  18713     1  ... python /home/fcm/cylc-6.11.4/bin/cylc-run␣
   ↪u-ak194
  1 S annette  18714 18713  ... python /home/fcm/cylc-6.11.4/bin/cylc-run␣
   ↪u-ak194
  1 S annette  18715 18713  ... python /home/fcm/cylc-6.11.4/bin/cylc-run␣
   ↪u-ak194
  1 S annette  18717 18713  ... python /home/fcm/cylc-6.11.4/bin/cylc-run␣
   ↪u-ak194
  1 S annette  18718 18713  ... python /home/fcm/cylc-6.11.4/bin/cylc-run␣
   ↪u-ak194
  ```

  This gives a list of processes. The number in the 4th column is the process-id. Use this to kill each of the processes, eg:

  ```
  kill -9 18713
  ```

- Delete the port file:

  This lives under `~/.cylc/ports/`. For example: `rm ~/.cylc/ports/u-ak194`

SECTION

ELEVEN

APPENDIX B: SSH FAQS

This Sections provides instructions for some common ssh tasks. If you have any problems, contact a member of the CMS team.

11.1 Using an existing ssh agent

If you already have an ssh-agent set up on PUMA, you can use this one to connect to your Archer training account. Conversely, after the course you may wish to use the keys you set up for own Archer account.

You can copy your ssh key over to Archer using the `ssh-copy-id` script.

First you need to find the name of the public key in your `.ssh` directory.

```
puma$ cd ~/.ssh
puma$ ls
environment.puma  id_rsa  id_rsa.pub  known_hosts  ssh-setup
```

The public key ends with `.pub` and will usually be called `id_rsa.pub` or `id_dsa.pub`.

Now run the script to copy the key to your Archer account, making sure to use the correct name for your key:

```
puma$ ssh-copy-id -i ~/.ssh/id_rsa.pub <archer-username>@login.archer.ac.uk
```

You will be prompted for your Archer password.

If successful, you should now be able to login to Archer without a password. If you are prompted for a passphrase you need to re-start your agent - see below.

11.2 Restarting your ssh agent

Normally your ssh agent persists even when you log out of puma. However, from time to time it can vanish.

If you are prompted for your passphrase, this means the ssh agent has stopped for some reason. The agent *should* have been re-initialised when you logged into puma, but you will need to re-associate your ssh keys to the agent.

To do so, run:

```
puma$ ssh-add
```

If successful this will prompt for your passphrase:

```
Enter passphrase for /home/<puma-username>/.ssh/id_rsa:
```

Sometimes this step will fail with the following error:

```
Could not open a connection to your authentication agent.
```

In this case, the agent is not running. Usually this is beacuse of an environment file. Delete the following:

```
puma$ rm ~/.ssh/environment.puma
```

Then log out of puma and back in again. You should hopefully see a message similar to:

```
Initialising new SSH agent...
```

And you should now be able to run ssh-add successfully.

11.3 Regenerating your ssh keys

If you have forgotten your passphrase you will need to regenerate your ssh keys. Before doing so, you will need to tidy up the old keys otherwise the ssh agent can get itself confused.

Go to your `.ssh` directory, and look at the files:

```
puma$ cd ~/.ssh
puma$ ls
environment.puma  id_rsa  id_rsa.pub  known_hosts  ssh-setup
```

Delete the public and private keys. These will normally be named `id_rsa` and `id_rsa.pub`, or `id_dsa` and `id_dsa.pub`.

You should also delete the `environment.puma` file:

```
puma$ rm id_rsa id_rsa.pub environment.puma
```

Next check if you have an agent running:

```
puma$ ps -flu <puma-username> | grep ssh-agent
```

If you have an agent running, one or more lines like the following will be returned:

```
15658 ?        00:00:00 ssh-agent
```

The number in the first column is the process-id, pass this to the `kill` command to stop the process, for example:

```
puma$ kill -9 15658
```

You can now start again, following the *ssh set up instructions*.