NorESM user workshop 2023

28 Nov. 2023

Steve Goldhaber, stevenng@met.no
Mariana Vertenstein, marianav@met.no

Agenda

Today, we try to incorporate feedback from the first two days of the workshop by discussing some of the earlier topics in more depth and introducing some topics that were mentioned or requested but were not part of the original agenda.

1. Reproducible experiment and model development
2. New infrastructure version of NorESM (beginning with NorESM 2.5)
3. The Spectral Element (SE) dycore for the NorESM atmosphere model (CAM-Nor)

Reproducible Experiment and Model Design

1. Compsets, grids, and cases (looking inside your case) (Mariana)
2. Reproducible development (scripts) (Steve)
3. Model development – changing the code (Steve)

Reproducible Development with Scripts

- Most aspects of reproducible experiments can be included in a simple bash script
- A script is a single file that can be shared and stored.
  - It addition to creating all the settings found in the case documentation, it can (should) also control the exact code version used for the experiment.
- Elements of a script
  - Make sure the model source (`SRCROOT`) has the correct version of the model (`TAG`) checked out (including infrastructure and component models)
  - Call **`create_newcase`** with a compset (`COMPSET`), model grid (`RES`), case (`CASEDIR`) and any other desired options
  - Execute any need **`xmlchange`** commands
  - Call **`case.setup`**
  - Add any desired custom namelist settings
  - Call **`case.build`** and **`case.submit`**
  - Check for errors in any of these steps

Sample Experiment Script (1)

```
#! /bin/bash
## Run an out-of-the-box N2000 experiment for 13 months
##    but with reduced output.

## Experiment basics, modify these for your experiment
TAG="release-noresm2.0.7"
COMPSET="N2000"
RES="f19_tn14"
SRCROOT="/cluster/projects/nn1001k/xxUSERxx/NorESM"
CASEDIR="/cluster/work/users/xxUSERxx/cases"
REPO="https://github.com/NorESMhub/NorESM"
PROJECT="nn9039k"

## (make sure that clone exists, otherwise, clone REPO)
if [ ! -d "${SRCROOT}" ]; then
    git clone -o NorESM ${REPO} ${SRCROOT}
fi

## Ensure correct source is checked out
cd ${SRCROOT}
git checkout ${TAG}
./manage_externals/checkout_externals
```

Sample Experiment Script (2)

```
## Create your case
./cime/scripts/create_newcase --case ${CASEDIR} --compset ${COMPSET} --res ${RES}

## Any PE changes must go here

## Changes that affect the build go here
./xmlchange DEBUG=TRUE
./xmlchange STOP_OPTION=nmonths,STOP_N=13

## Build the model
./case.build

## Last chance to modify run-time settings
echo "history_chemistry       = .false." >> user_nl_cam
echo "history_chemspecies_srf = .false." >> user_nl_cam
echo "history_clubb           = .false." >> user_nl_cam

## Submit the job
./case.submit
```

Model Development – Working with the code

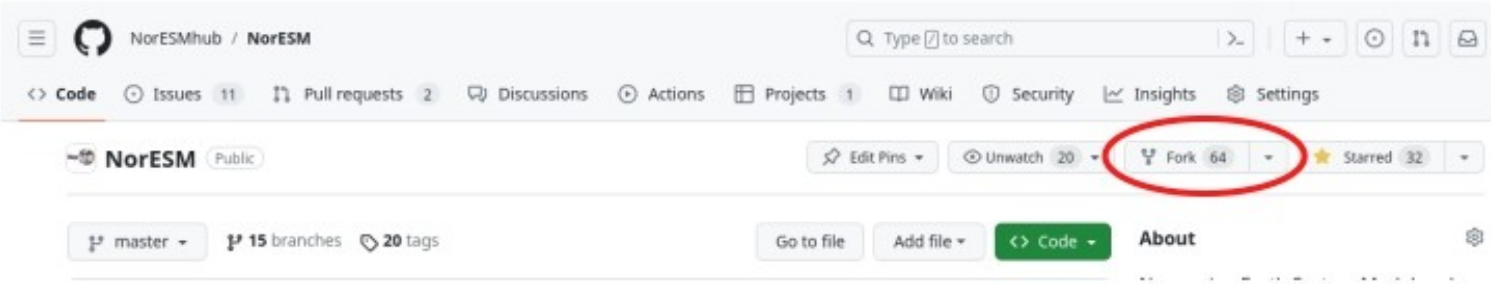
Steps to working with code modifications using git and GitHub

1. Create a personal fork of the repository where you want to make changes*
2. Add that fork as a new remote in your clone*
3. Create and checkout a branch to store your changes
4. Make any changes to your branch and run tests (experiments)
5. Commit early and often to make it easy to track what was changed when
6. Push your branch to your fork for backup and sharing

* One time set-up

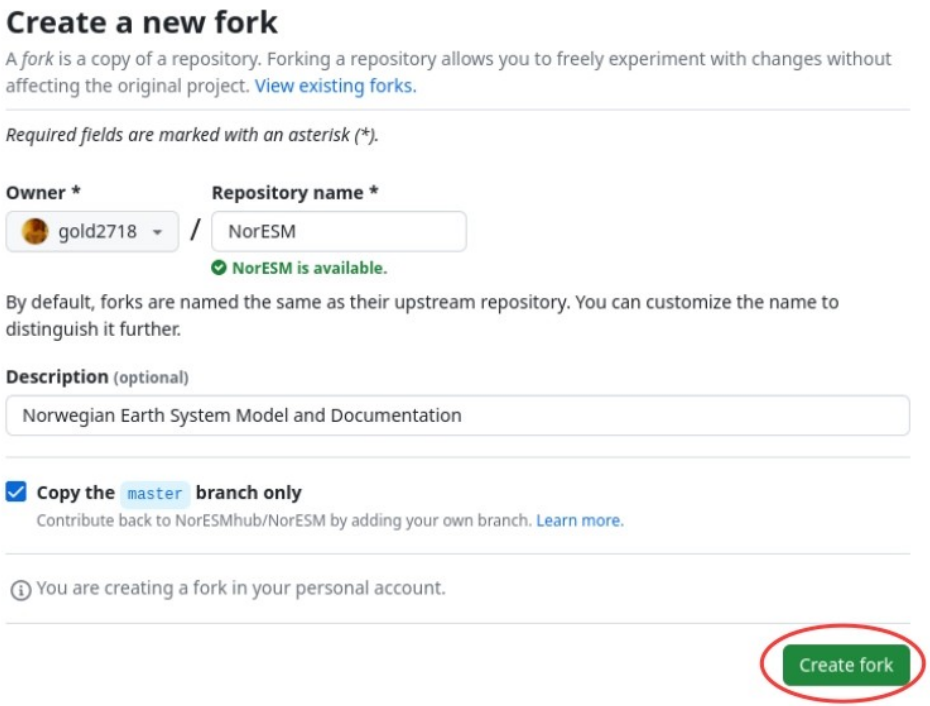
Working with the code – create a personal fork

- A “fork” is a GitHub term for a repository that remembers where it came from.
- A fork is a standard GitHub repository with the added feature that you can see the other forks and open a Pull Request (PR) to one of them.
- From the main page of NorESMhub/NorESM (or NorESMhub/<component>):



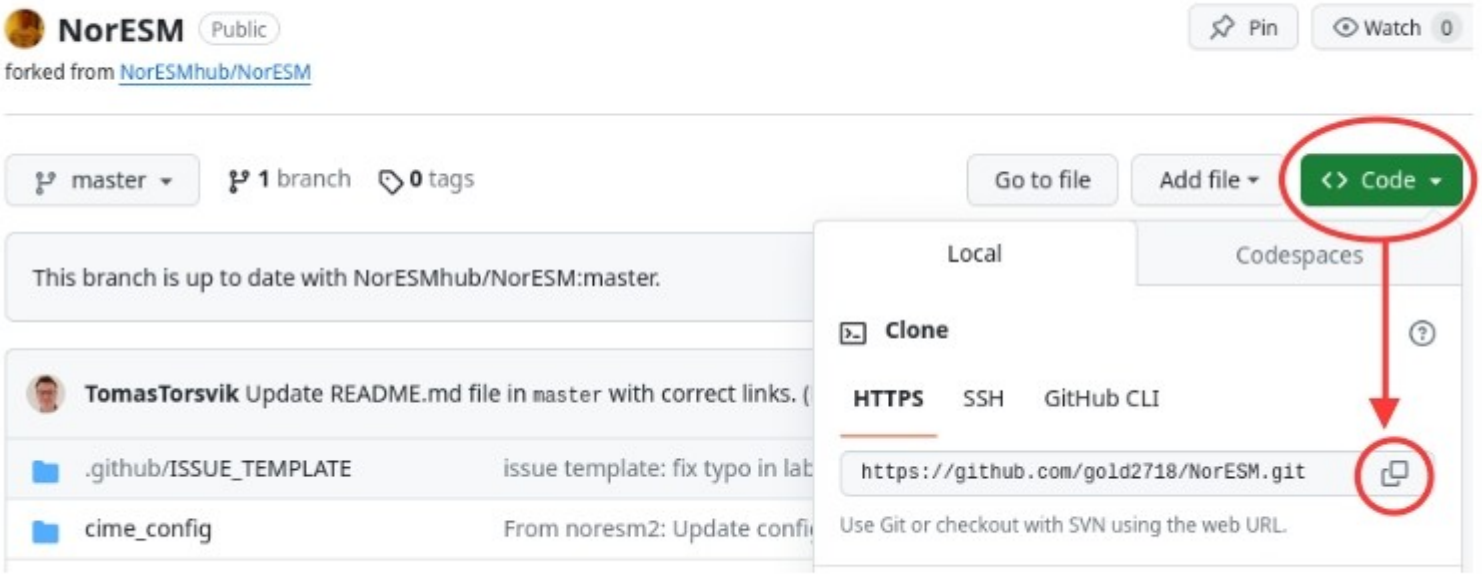
Working with the code – create a personal fork

- After you are satisfied with with the settings, press the “Create Fork” button.



Working with the code – adding a new remote

- Copy the URL for your new fork



Working with the code – adding a new remote (cont.)

- Copy the URL for your new fork

```
$ git remote add <name> https://github.com/<name>/NorESM.git

$ git remote -v

origin https://github.com/NorESMhub/NorESM (fetch)
origin https://github.com/NorESMhub/NorESM (push)
<name> https://github.com/<name>/NorESM.git (fetch)
<name> https://github.com/<name>/NorESM.git (push)
```

Working with the code – creating a new branch

- A branch in git is just a new pointer to the last commit.
- When creating a new branch, always declare where the branch will begin

```
$ git branch my_edit release-noresm2.0.7
```

```
$ git checkout my_edit
```

- You can combine the new branch creation and checkout

```
$ git checkout -b my_edit release-noresm2.0.7
```

Working with the code – making code changes

- If you make changes to source code, you need to call **`./case.build`** again
- If you change any Fortran **`use`** statements, you should rebuild the model:

`./case.build –clean <component>; ./case.build`

- where **`<component>`** is a component type such as **atm** or **ocn**.
- The fastest way to force a rebuild is: **`rm -r bld`**
- If you change anything in **cime_config**, you should start with a new case

Working with the code – using commits as a development tool

- Make sure your git environment is configured correctly
- For any machine you work on (e.g., Betzy), you need to have these settings:

```
git config --global user.name "Your Name"

git config --global user.email <GitHub email address>
```

- Useful introduction to git from Software Carpentry: https://swcarpentry.github.io/git-novice/
- Also, see the “contribute” section in the NorESM documentation
- If you like looking at code differences graphically, you can try git’s difftool:

```
git config --global diff.tool=meld

git config --global difftool.prompt=false
```

Working with the code – using commits as a development tool

- Commit often! Commits help you track differences and find problems.
- Always enter a useful log message, future you will thank you.
- If you create a new source file, be sure to add it before making your commit:

```
$ git add <new_file>
```

- To commit all changes and type in a log message:

```
$ git commit -am ‘Great log message here’
```

- To commit all changes and have an editor window open to enter your log message:

```
$ git commit -a
```

Working with the code – interacting with your fork

- Push your branch to your fork. This serves both as a backup and as a means to share – with others or with yourself on other machines.
- Always specify the destination repository and the branch that you want to push.

```
git push <name> my_edit
```

Working with the code – updating Externals.cfg

- If you are making edits to a branch of a component model and want to perform coupled runs with NorESM, you need to update the Externals.cfg file in NorESM. (Note that for most component models (e.g., CAM, CTSM), you can build and run non-coupled cases with just a clone of that component.)
- Make a branch of your NorESM clone and edit Externals.cfg (example below for CAM):

```
[cam]
tag = cam_cesm2_1_rel_05-Nor_v1.0.5
protocol = git
repo_url = https://github.com/NorESMhub/CAM
local_path = components/cam
required = True
```

```
[cam]
branch = my_edit
protocol = git
repo_url = https://github.com/<name>/CAM
local_path = components/cam
required = True
```

Working with the code – finding code differences

- git diff functionality is a great tool for finding bugs.
- Find all the files that changed between two commits
  `git diff –name-only <commit1> <commit2>`
- **`<commit1>`** can be a hash, branch name, or tag. If you leave off **`<commit2>`**, the current state will be used.
  `git diff <commit1> <commit2> [ -- path/to/file.F90 ]`
- If you leave off the **`-- path/to/file.F90`**, the difference for all files will be shown.
- **`git difftool`** shows a graphical version of file differences
  `git difftool <commit1> <commit2> [ -- path/to/file.F90 ]`
- Find differences from the last commit.
  `git diff(tool) HEAD^1 [ -- path/to/file.F90 ]`

Agenda

Today, we try to incorporate feedback from the first two days of the workshop by discussing some of the earlier topics in more depth and introducing some topics that were mentioned or requested but were not part of the original agenda.

1. Reproducible experiment and model development
2. New infrastructure version of NorESM (beginning with NorESM 2.5)
3. The Spectral Element (SE) dycore for the NorESM atmosphere model (CAM-Nor)

Agenda

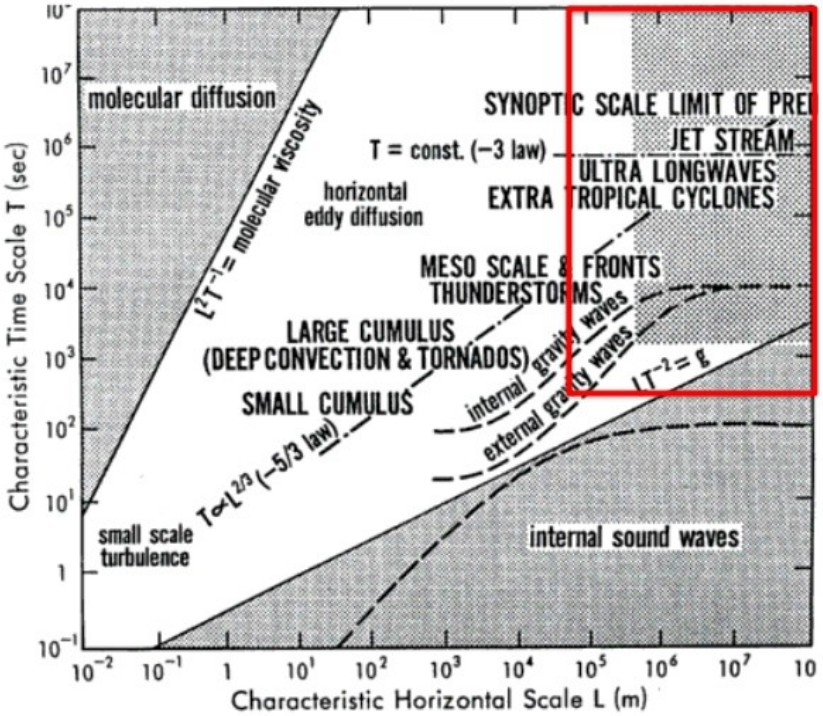
Today, we try to incorporate feedback from the first two days of the workshop by discussing some of the earlier topics in more depth and introducing some topics that were mentioned or requested but were not part of the original agenda.

1. Reproducible experiment and model development
2. New infrastructure version of NorESM (beginning with NorESM 2.5)
3. The Spectral Element (SE) dycore for the NorESM atmosphere model (CAM-Nor)

The Spectral Element (SE) Atmosphere Dynamical Core (dycore)

1. What role does the dycore play in the NorESM atmosphere model?
2. What is the SE dycore
3. Why do we need a new dycore?

Spatial and temporal scales of the atmosphere



- The recorded resolves the large scale movement of the atmosphere by solving the Navier-Stokes equations discretized onto a grid.
- Smaller scale (both time and space) phenomena are simulated by parameterizations.

Courtesy: Smagorinsky (1974).

The Finite Volume (FV) and Spectral Element (SE) dycores

CAM finite volume

CAM spectral element

The FV grid is a traditional latitude / longitude (rectangular) grid. The SE grid is a cubed-sphere with each face broken into spectral elements.

Why do we need a new dycore?

- The FV dycore does not scale well to large numbers of MPI tasks.
- The SE dycore has better scaling to large numbers of MPI tasks and for regular (non-refined) grids, also scales quite well in the number of chemical tracers.
- The FV dycore is “end of life” in that it is no longer being developed.
- New capabilities related to energy and mass conservation are very important for future versions of NorESM. The SE dycore is being developed with these capabilities.
- The SE dycore supports refined grids which enable detailed studies over particular regions of interest.