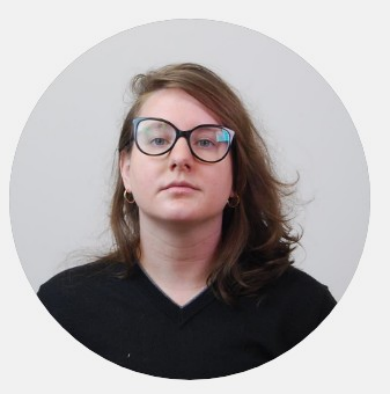
Beyond Inlists

MESA

Summer School 2022

Lecture 1



Dr Meridith Joyce

Lasker Fellow, Space Telescope Science Institute

Incoming: Marie Curie Widening Fellow, Konkoly Observatory

MESA developers team (2019-present)

@MeridithJoyceGR

www.meridithjoyce.com

Environment Variables

Environment variables
To run MESA, you must set *environment variables* that point your system to the correct paths for dependent packages

Environment Variables

Environment variables
To run MESA, you must set *environment variables* that point your system to the correct paths for dependent packages

The variables are:

MESA_DIR – the location of your top-level MESA directory

MESASDK_ROOT – the location of the SDK

OMP_NUM_THREADS – the number of threads (= 2x number of cores) you choose to devote to MESA calculations.

Environment Variables

Environment variables
To run MESA, you must set *environment variables* that point your system to the correct paths for dependent packages

The variables are:

MESA_DIR – the location of your top-level MESA directory

MESASDK_ROOT – the location of the SDK

OMP_NUM_THREADS – the number of threads (= 2x number of cores) you choose to devote to MESA calculations.

Note that most laptops have between 2 and 8 threads available, or 1 to 4 cores. Setting **OMP_NUM_THREADS** above this will result in overextending the capabilities of your system, causing the calculations to run very slowly or crash

Environment Variables

Environment variables
To run MESA, you must set *environment variables* that point your system to the correct paths for dependent packages

The variables are:

MESA_DIR – the location of your top-level MESA directory

MESASDK_ROOT – the location of the SDK

OMP_NUM_THREADS – the number of threads (= 2x number of cores) you choose to devote to MESA calculations.

Note that most laptops have between 2 and 8 threads available, or 1 to 4 cores. Setting **OMP_NUM_THREADS** above this will result in overextending the capabilities of your system, causing the calculations to run very slowly or crash

The syntax for setting these variables differs between MacOS and Linux and depends on your shell environment (ex. I use Linux and bash shell)

Environment Variables

Environment variables
To run MESA, you must set *environment variables* that point your system to the correct paths for dependent packages

The variables are:

MESA_DIR – the location of your top-level MESA directory

MESASDK_ROOT – the location of the SDK

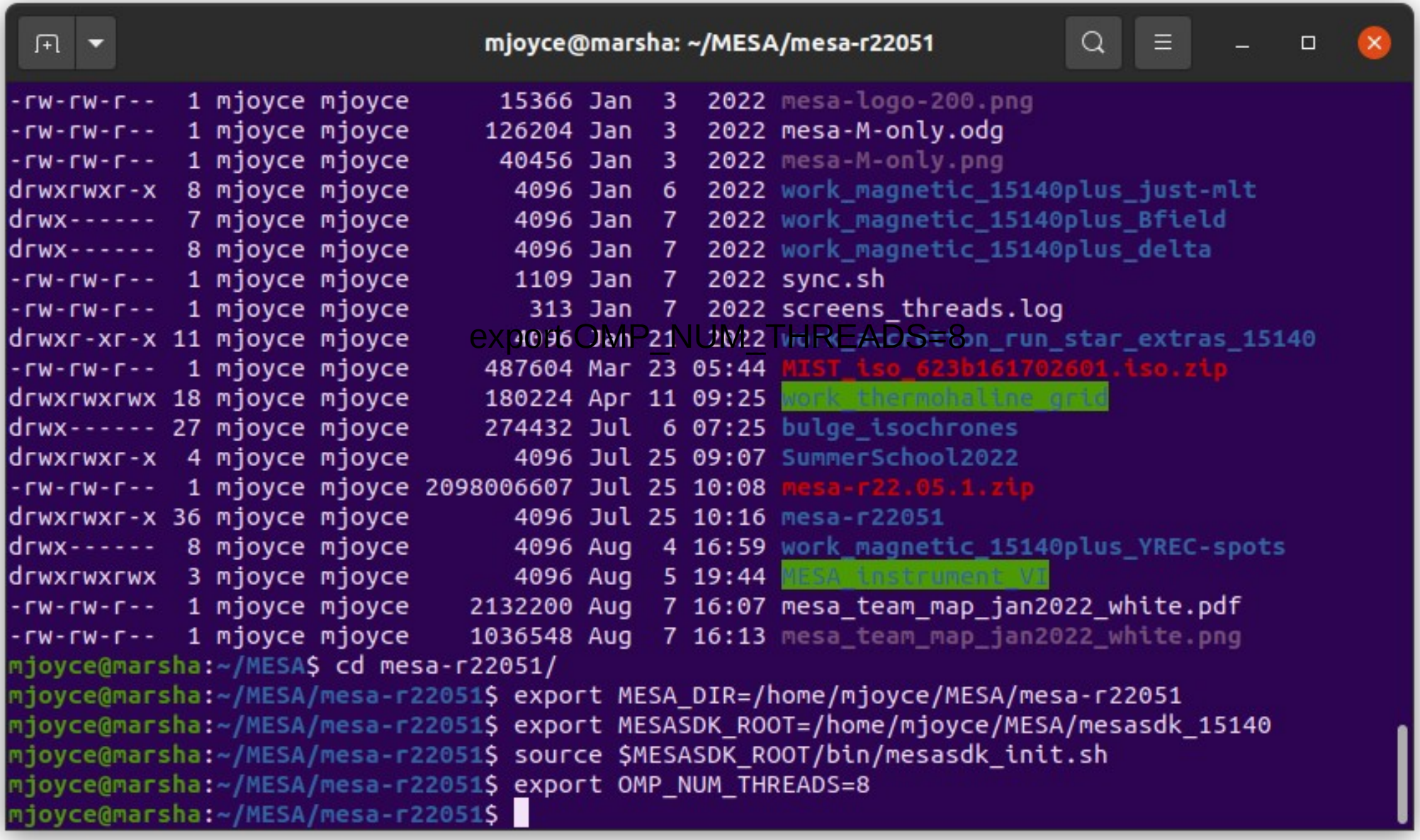
OMP_NUM_THREADS – the number of threads (= 2x number of cores) you choose to devote to MESA calculations.

Note that most laptops have between 2 and 8 threads available, or 1 to 4 cores. Setting **OMP_NUM_THREADS** above this will result in overextending the capabilities of your system, causing the calculations to run very slowly or crash

The syntax for setting these variables differs between MacOS and Linux and depends on your shell environment (ex. I use Linux and bash shell)

Visit http://cococubed.asu.edu/mesa_market/guides.html for video guides on how to set up these variables for other systems

Setting environment variables the bad way

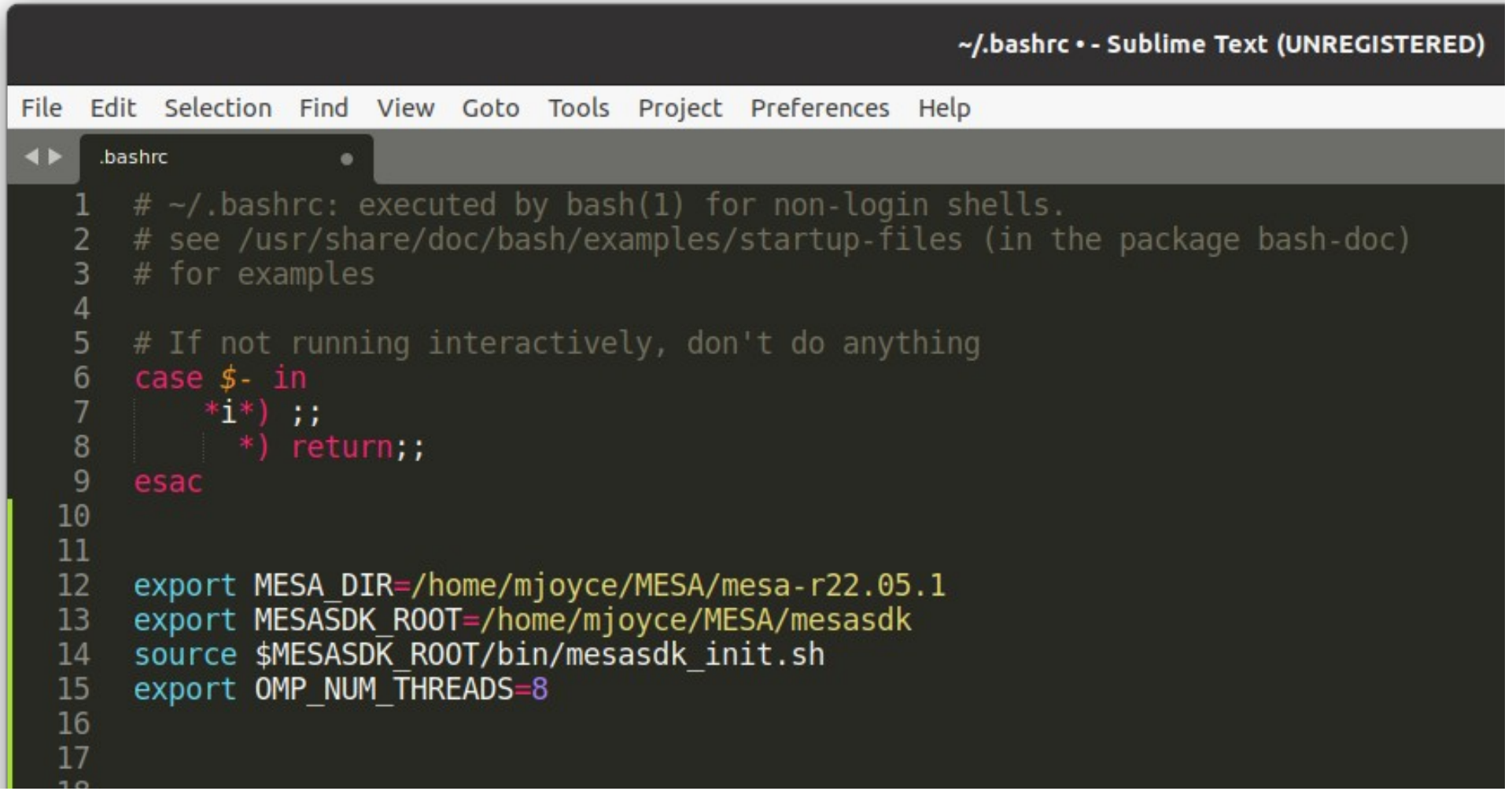


Setting environment variables the good way: .bashrc file

Since **.bashrc** (or equivalent) is read automatically each time you open a new terminal window, assigning environment variables in **.bashrc** means you do not need to assign them manually

Setting environment variables the good way: .bashrc file

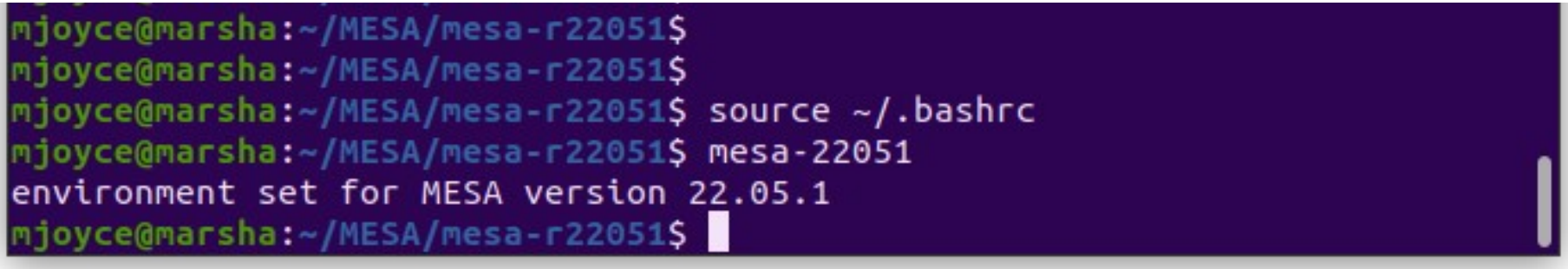
Since **.bashrc** (or equivalent) is read automatically each time you open a new terminal window, assigning environment variables in **.bashrc** means you do not need to assign them manually



Best way: functions in .bashrc

```
354
355  function mesa-12778 {
356  export MESA_DIR=/home/mjoyce/MESA/mesa-r12778
357  export MESASDK_ROOT=/home/mjoyce/MESA/mesasdk_12778
358  source $MESASDK_ROOT/bin/mesasdk_init.sh
359  export OMP_NUM_THREADS=8
360  echo "environment set for MESA version 12778"
361  }
362
363
364  function mesa-15140 {
365  export MESA_DIR=/home/mjoyce/MESA/mesa-r15140
366  export MESASDK_ROOT=/home/mjoyce/MESA/mesasdk_15140
367  source $MESASDK_ROOT/bin/mesasdk_init.sh
368  export OMP_NUM_THREADS=8
369  echo "environment set for MESA version 15140"
370  }
371
372
373  function mesa-22051 {
374  export MESA_DIR=/home/mjoyce/MESA/mesa-r22.05.1
375  export MESASDK_ROOT=/home/mjoyce/MESA/mesasdk
376  source $MESASDK_ROOT/bin/mesasdk_init.sh
377  export OMP_NUM_THREADS=8
378  echo "environment set for MESA version 22.05.1"
379  }
380
```

Best way: functions in .bashrc



Do microlab 0

(~3 minutes)

Core Resources

The code itself:
Mesa-r22.05.1/star/test_suite
Mesa-r22.05.1/star/defaults/*.list ; *.defaults

The web-hosted documentation:
https://docs.mesastar.org/en/release-r22.05.1/

inlists used in academic papers:
http://cococubed.asu.edu/mesa_market/inlists.html

Past MESA Summer School lectures and labs (including solutions):
http://cococubed.asu.edu/mesa_market/education.html

Mesa-users email list:
https://lists.mesastar.org/mailman/listinfo/mesa-users

py_mesa_reader by Bill Wolf:
https://github.com/wmwolf/py_mesa_reader

Setup and output

Inlists – Fortran namelists that contain **value** definitions for all of the **parameters** of your run

ex) history_filename = ‘history_my_run.data’

Setup and output

Inlists – Fortran namelists that contain **value** definitions for all of the **parameters** of your run

ex) history_filename = ‘history_my_run.data’

Parameter libraries- *all of the possible values for your parameters can be found in the module defaults files*

Setup and output

Inlists – Fortran namelists that contain **value** definitions for all of the **parameters** of your run

ex) history_filename = ‘history_my_run.data’

Parameter libraries- *all of the possible values for your parameters can be found in the module defaults files*

mesa-r22.05.1/star/defaults/ contains

controls.defaults
star_job.defaults
pgstar.defaults
history_columns.list
profile_columns.list

Setup and output

Inlists – Fortran namelists that contain **value** definitions for all of the **parameters** of your run

ex) history_filename = ‘history_my_run.data’

Parameter libraries- *all of the possible values for your parameters can be found in the module defaults files*

mesa-r22.05.1/star/defaults/ contains

- controls.defaults
- star_job.defaults
- pgstar.defaults
- history_columns.list
- profile_columns.list

Opacity defaults can be found in mesa-r22.05.1/kap/defaults/kap.defaults

EOS defaults in mesa-r22.05.1/eos/defaults/eos.defaults

Setup and output

Inlists – Fortran namelists that contain **value** definitions for all of the **parameters** of your run

ex) history_filename = ‘history_my_run.data’

Parameter libraries- *all of the possible values for your parameters can be found in the module defaults files*

mesa-r22.05.1/star/defaults/ contains

- controls.defaults
- star_job.defaults
- pgstar.defaults
- history_columns.list
- profile_columns.list

Opacity defaults can be found in mesa-r22.05.1/kap/defaults/kap.defaults

EOS defaults in mesa-r22.05.1/eos/defaults/eos.defaults

Executable

star or **binary**; this is the program that is built by the compiler and runs your simulation

Setup and output

Inlists – Fortran namelists that contain **value** definitions for all of the **parameters** of your run

ex) history_filename = ‘history_my_run.data’

Parameter libraries- *all of the possible values for your parameters can be found in the module defaults files*

mesa-r22.05.1/star/defaults/ contains

- controls.defaults
- star_job.defaults
- pgstar.defaults
- history_columns.list
- profile_columns.list

Opacity defaults can be found in mesa-r22.05.1/kap/defaults/kap.defaults

EOS defaults in mesa-r22.05.1/eos/defaults/eos.defaults

Executable

star or **binary**; this is the program that is built by the compiler and runs your simulation

Scripts

clean, mk, rn, re – these are shell scripts that build and manipulate your program

Setup and output

By default, MESA keeps track of the full stellar structure of your model across evolutionary time

Output is stored in the LOGS/ directory

history.data traces evolutionary quantities

profileX.data gives you the structural model at some time step *dt.* you can adjust the frequency of these outputs in the inlists

profiles.index provides a mapping between the integer in the profile output names and the model number from the evolutionary run (in cases where a profile is not generated at every time step)

You can also store binary snapshots of the models: **photos**

Setup and output

History output should look something like this:

~/MESA/mesa-r15140/star/work/LOGS/history.data - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

inlist_start_header | inlist_start | .bashrc | history.data

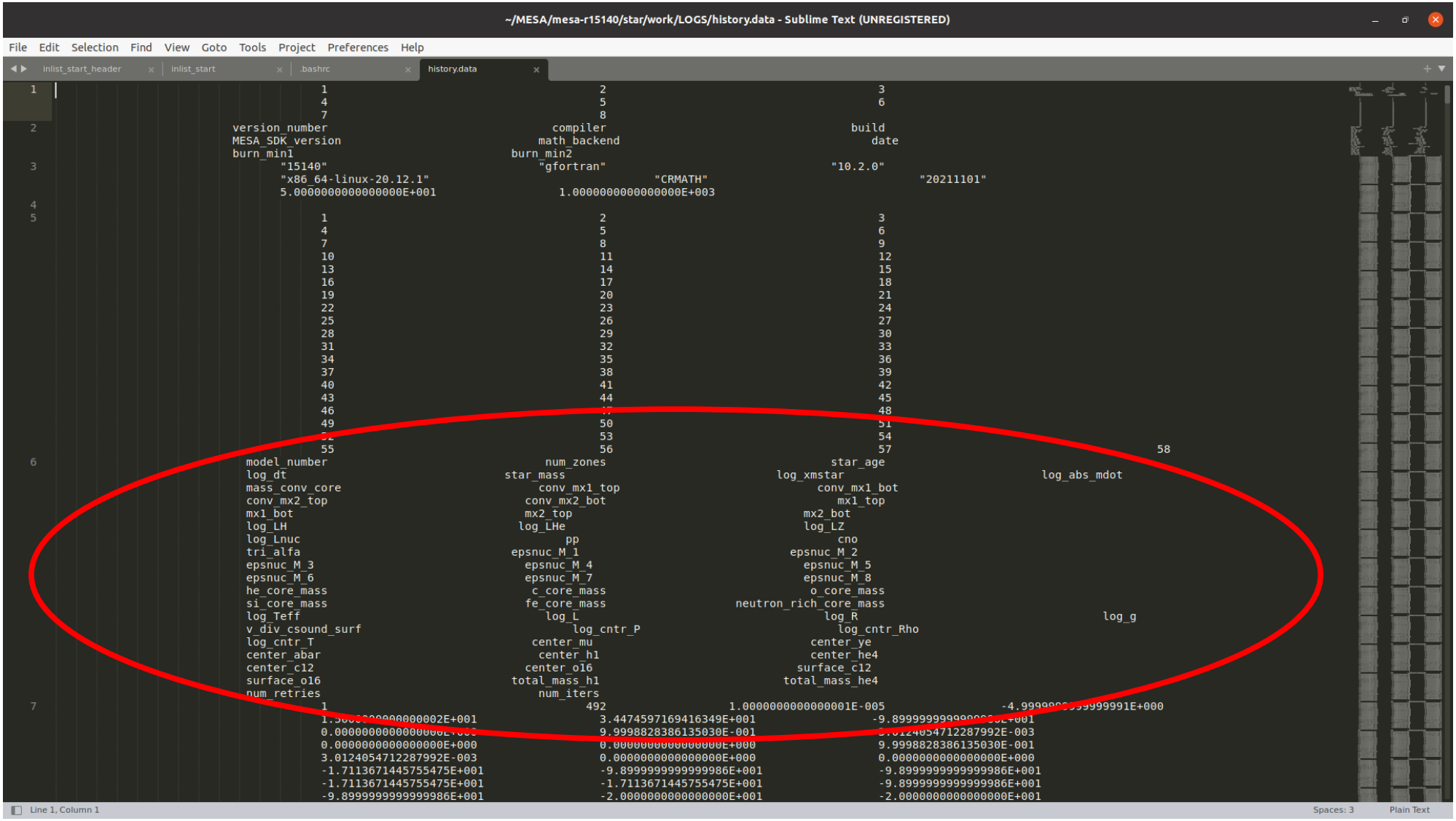
```
                      1                   2                   3
                      4                   5                   6
                      7                   8
version_number                compiler                build
MESA_SDK_version              math_backend            date
burn_min1                     burn_min2
      "15140"                 "gfortran"              "10.2.0"
      "x86_64-linux-20.12.1"                "CRMATH"                "20211101"
      5.0000000000000000E+001           1.0000000000000000E+003

                      1                   2                   3
                      4                   5                   6
                      7                   8                   9
                     10                  11                  12
                     13                  14                  15
                     16                  17                  18
                     19                  20                  21
                     22                  23                  24
                     25                  26                  27
                     28                  29                  30
                     31                  32                  33
                     34                  35                  36
                     37                  38                  39
                     40                  41                  42
                     43                  44                  45
                     46                  47                  48
                     49                  50                  51
                     52                  53                  54
                     55                  56                  57                  58
 model_number              num_zones               star_age
 log_dt                    star_mass               log_xmstar              log_abs_mdot
 mass_conv_core            conv_mx1_top            conv_mx1_bot
 conv_mx2_top              conv_mx2_bot            mx1_top
 mx1_bot                   mx2_top                 mx2_bot
 log_LH                    log_LHe                 log_LZ
 log_Lnuc                  pp                      cno
 tri_alfa                  epsnuc_M_1              epsnuc_M_2
 epsnuc_M_3                epsnuc_M_4              epsnuc_M_5
 epsnuc_M_6                epsnuc_M_7              epsnuc_M_8
 he_core_mass              c_core_mass             o_core_mass
 si_core_mass              fe_core_mass            neutron_rich_core_mass
 log_Teff                  log_L                   log_R                   log_g
 v_div_csound_surf         log_cntr_P              log_cntr_Rho
 log_cntr_T                center_mu               center_ye
 center_abar               center_h1               center_he4
 center_c12                center_o16              surface_c12
 surface_o16               total_mass_h1           total_mass_he4
 num_retries               num_iters
                      1                 492   1.0000000000000001E-005  -4.9999999999999991E+000
    1.5000000000000002E+001   3.4474597169416349E+001  -9.8999999999999986E+001
    0.0000000000000000E+000   9.9998828386135030E-001   3.0124054712287992E-003
    0.0000000000000000E+000   0.0000000000000000E+000   9.9998828386135030E-001
    3.0124054712287992E-003   0.0000000000000000E+000   0.0000000000000000E+000
   -1.7113671445755475E+001  -9.8999999999999986E+001  -9.8999999999999986E+001
   -1.7113671445755475E+001  -1.7113671445755475E+001  -9.8999999999999986E+001
   -9.8999999999999986E+001  -2.0000000000000000E+001  -2.0000000000000000E+001
```

Line 1, Column 1 — Spaces: 3 — Plain Text

Setup and output

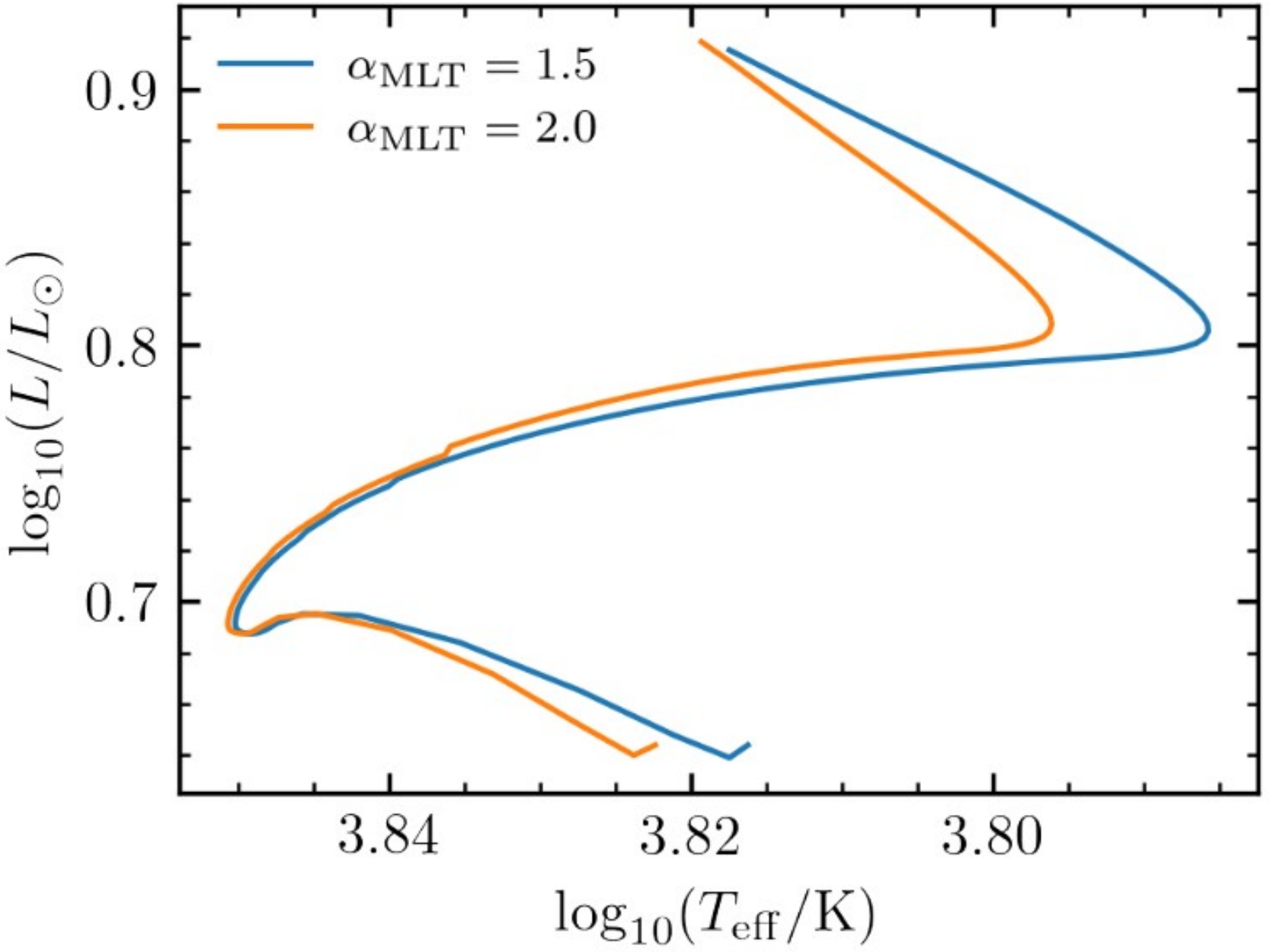
History output should look something like this:



Do up through Minilab 2

Reconvene for discussion at 10:25 (~25 minutes)

Discussion up through Minilab 2



Coffee break

Return at 10:45

Using run_star_extras.f90

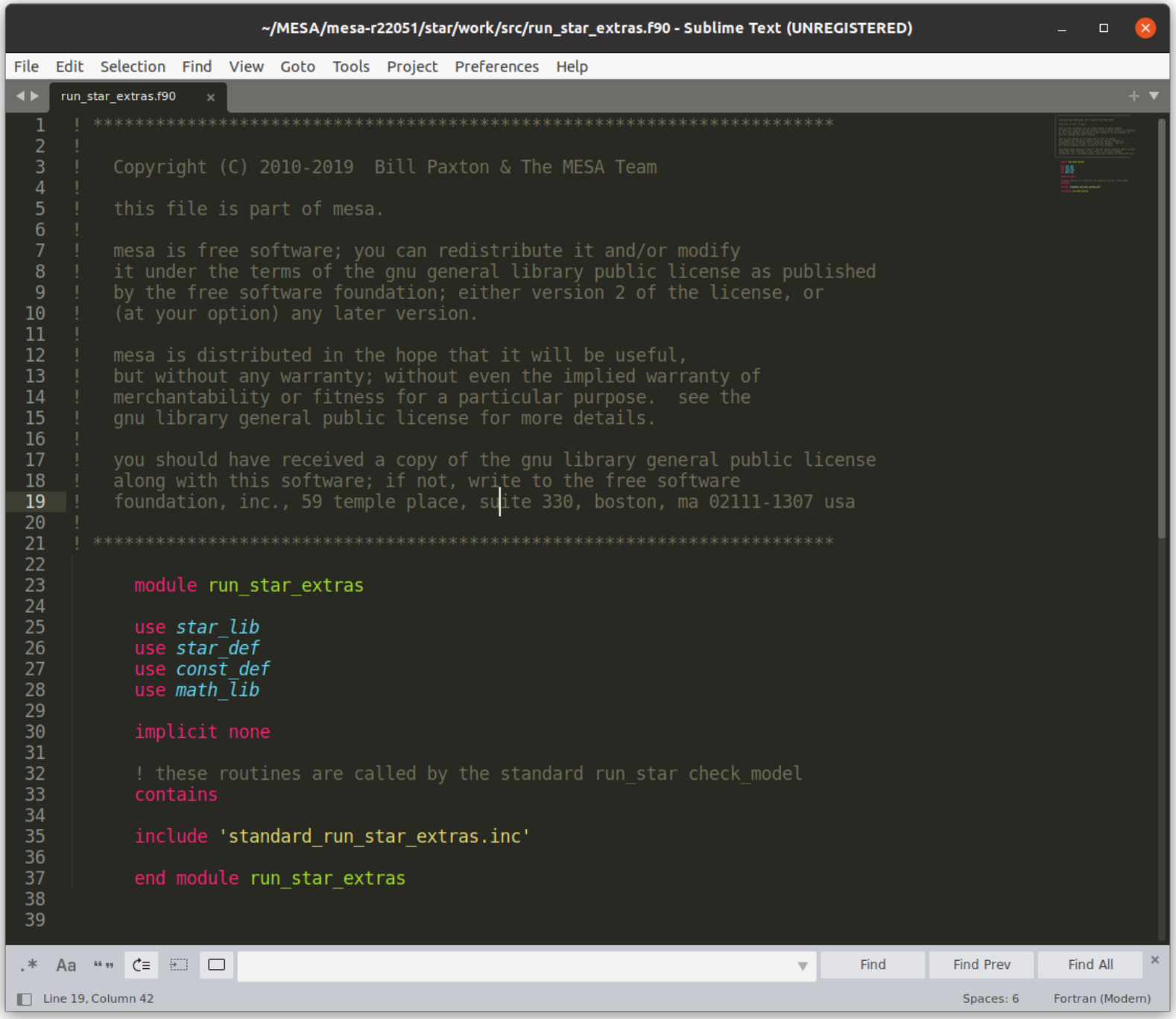
Using run_star_extras.f90, we can introduce our own

-**project-specific physics,** or

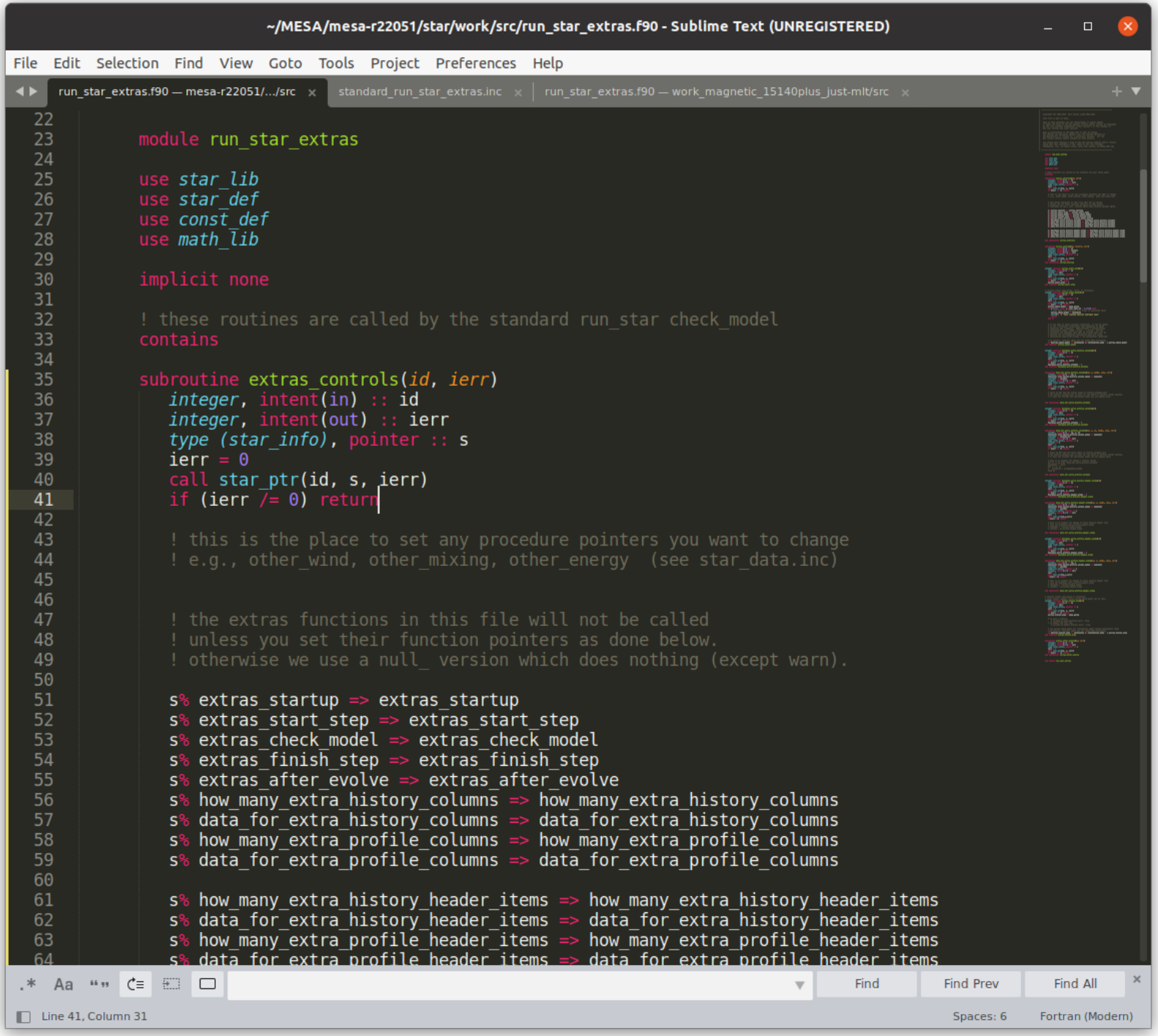
-additional functionality

without compromising the entire MESA source code base

run_star_extras.f90: default

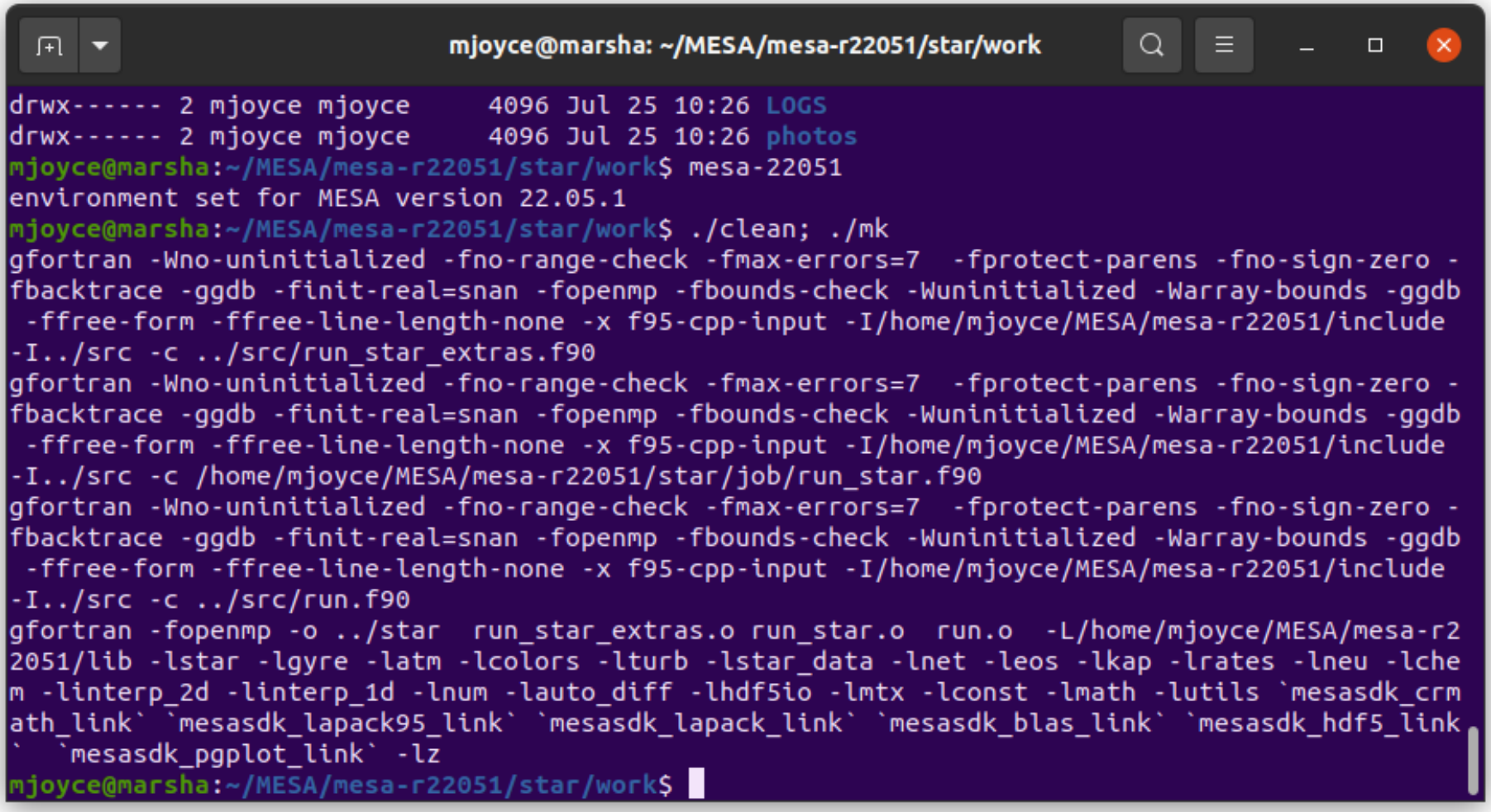


run_star_extras.f90: include

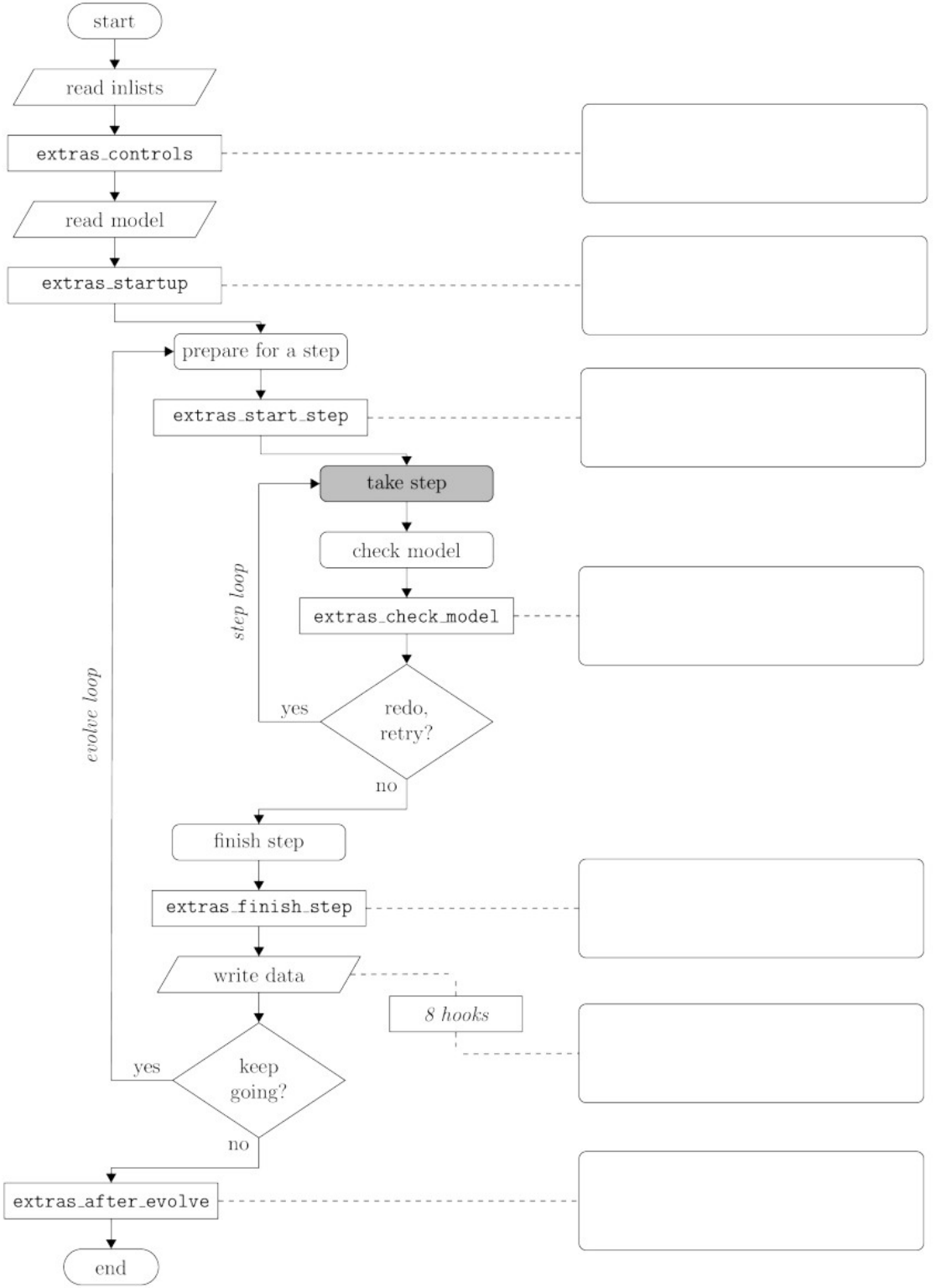


Using run_star_extras.f90

Every time you modify run_star_extras, you must **recompile the executable**



Code Organization



Code Organization

There are some actions you will want to compute *once per evolutionary time step* (**evolve loop**)

there are others you may want to compute *once per solver iteration* (**step loop**)

where one evolve step contains several solver iterations

Code Organization

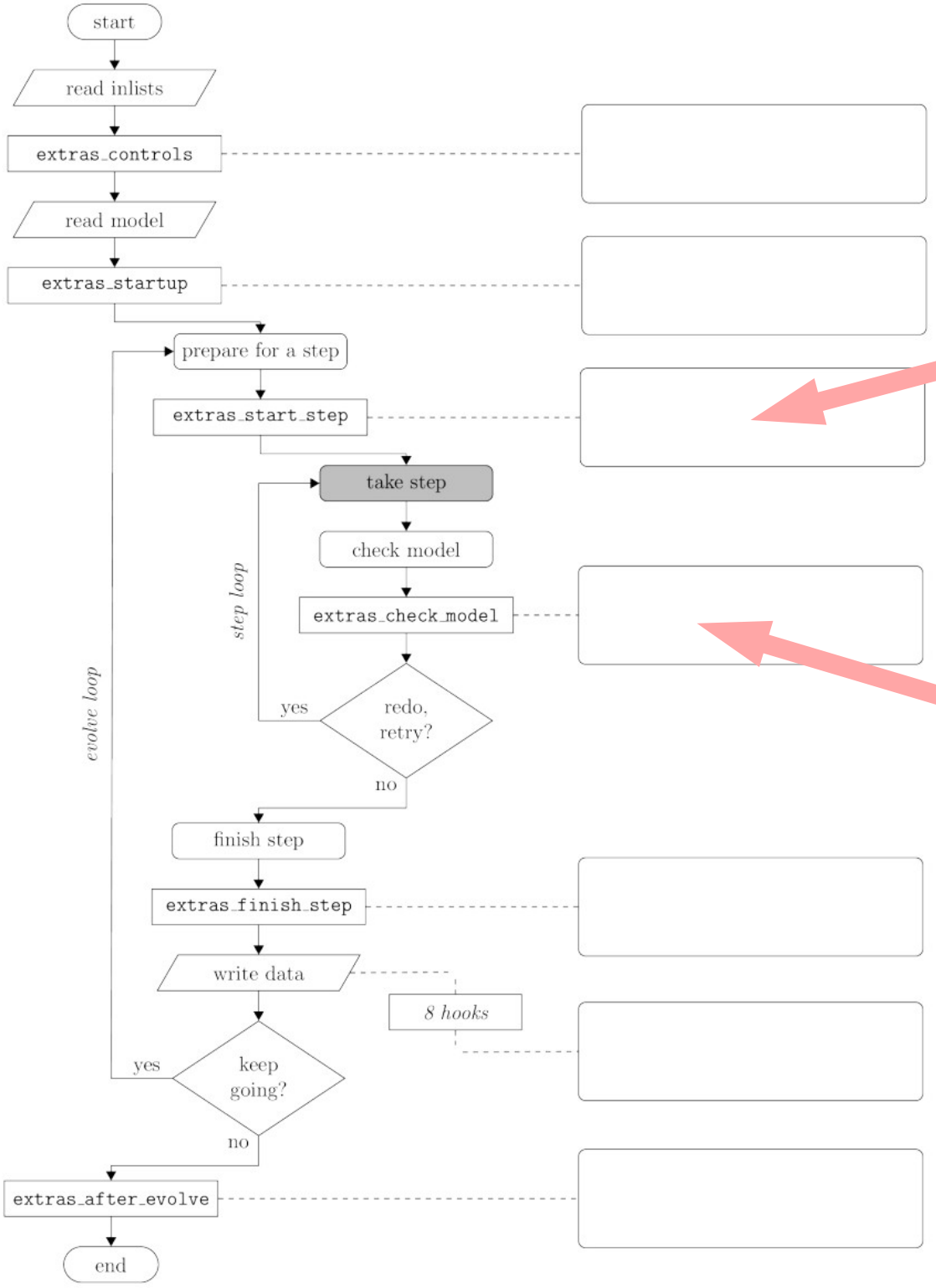
There are some actions you will want to compute *once per evolutionary time step* (**evolve loop**)

there are others you may want to compute *once per solver iteration* (**step loop**)

where one evolve step contains several solver iterations

Example:

Checking whether your model has satisfied some global physical property (i.e., reaching a certain radius) can take place once per evolutionary step




```
integer function extras_start_step(id)
   integer, intent(in) :: id
   integer :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   extras_start_step = 0
end function extras_start_step
```

```
! returns either keep_going, retry, or terminate.
integer function extras_check_model(id)
   integer, intent(in) :: id
   integer :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   extras_check_model = keep_going
   if (.false. .and. s% star_mass_h1 < 0.35d0) then
      ! stop when star hydrogen mass drops to specified level
      extras_check_model = terminate
      write(*, *) 'have reached desired hydrogen mass'
      return
   end if

   if (extras_check_model == terminate) s% termination_code = t_extras_check_model
end function extras_check_model
```

Suppose we want MESA to stop when the star reaches a certain luminosity.

-When during the step should this condition be checked?

-How often should this condition be checked?

-In which subroutine should we check this condition?

Do Maxilab 1

(~25 minutes)

Do Maxilab 2

(time remaining)