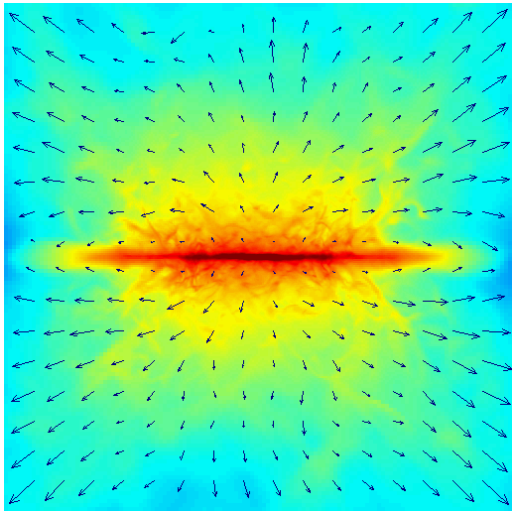
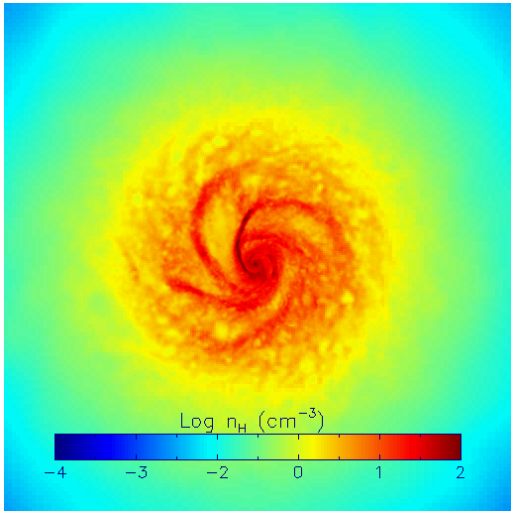
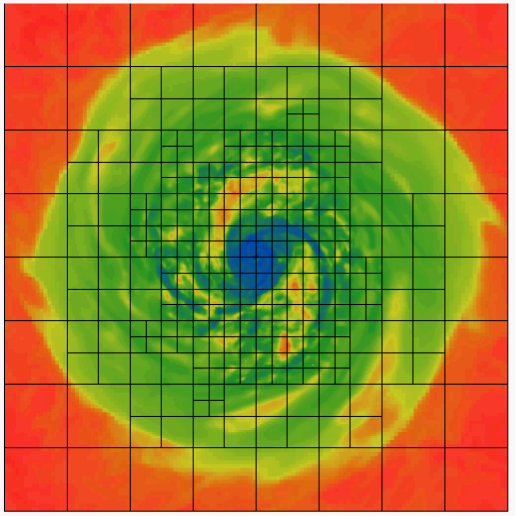
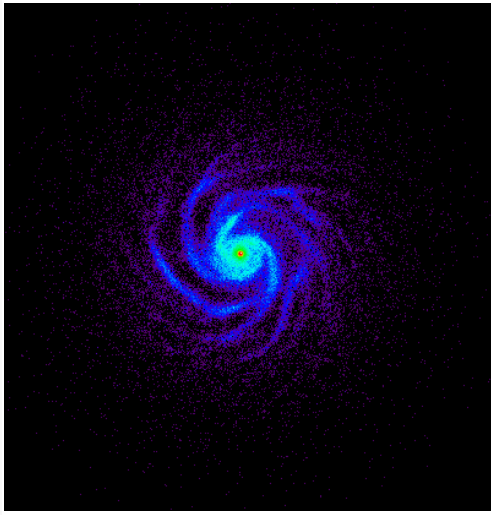
Numerical Practical in Astronomy

Visualisation

Nigel Mitchell

IDL – **I**nteractive **D**ata **L**anguage

- Interpreter language – you type commands and they are executed instantly.
- Licenses are very expensive.
- Free 7 minute timed demo mode with limited functionality: www.ittvis.com
- Inbuilt documentation is not ideal for beginners.
- See David Fanning's webpage or book for help,

  Web: www.dfanning.com

  Literature: IDL Programming Techniques, David Fanning
- Very widely used in research - particularly for observational astronomy
- Large amount of pre-written software available to download off the web.
- Usefull collection of some IDL `.pro` files can be downloaded from the course webpage:

  http://homepage.univie.ac.at/nigel.mitchell/NumPrac/

Different types of code:

Low Level of Abstraction

1. Machine code → Binary: 100101001
2. Compiled Languages, e.g. FORTRAN or C
   - User writes instructions using set syntax
   - Compiler translates syntax into machine code before program can be run.
   - Compilers can perform sophisticated optimisations.
3. Interpreted languages, e.g. IDL, python.
   - Code executes in real time as you type commands at a command line.
   - Easier and faster for development
   - Absence of compiler optimisation make execution slower than compiled languages

High Level of Abstraction

IDL

labserv (IDL 7.1):

```
> ssh -Y student1@labserv.astro...
> idl
```

lab machines 1-12 (IDL 7.0);

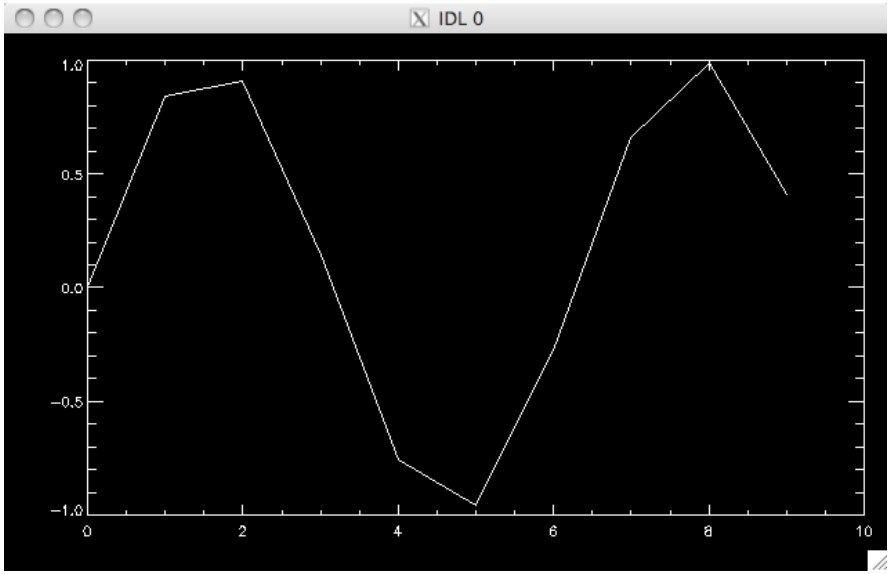
```
> ssh -Y student1@lab1.astro...
> idl
```

First Plots:

```
IDL> plot, x, sin(x)
% SIN: Variable is undefined: X
% Execution halted at: $MAIN$
IDL>
```

x has to be defined!

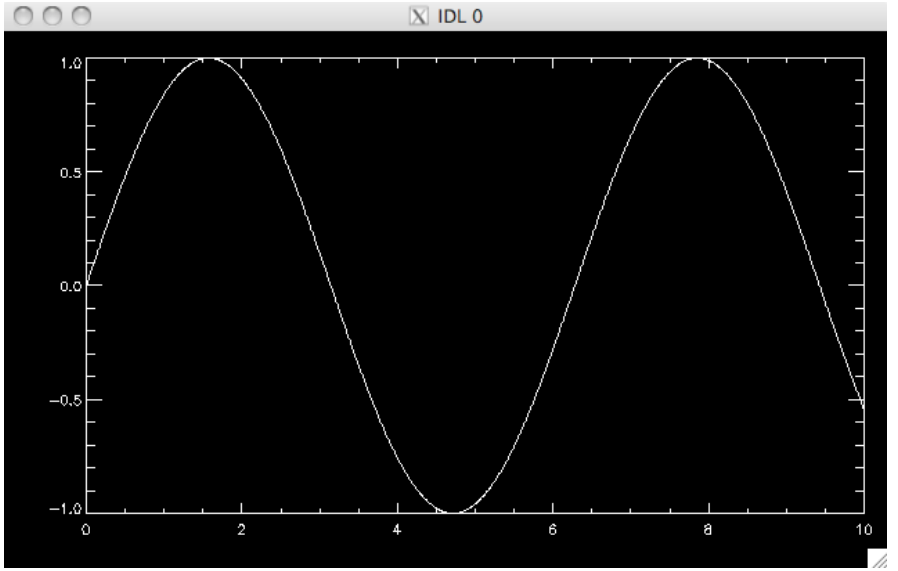
```
IDL> x = findgen (10)
IDL> plot, x, sin(x)
IDL>
```



IDL

More datapoints:

```
IDL> x = findgen (1000)/100.
IDL> plot, x, sin(x)
IDL>
```



Information about variables:

```
IDL> x = findgen (2)
IDL> help, x
X               FLOAT   = Array[2]
IDL> print, x
      0.00000     1.00000
```

Exit IDL:

```
IDL> exit
```

IDL

Save a session / running imported libraries

1.) Create a directory for your IDL programs

2.) Set the IDL_PATH for this directory in file *~/.bashrc*

```
export IDL_PATH=$IDL_PATH:
+/scratch/student1/NumPrak/idl_progs
```

3.) Write an IDL program in your IDL path

```
/scratch/student1/NumPrak/idl_progs/test1.pro
```

4.) Start IDL

5.) Compile the program

```
IDL> .compile test1.pro
```

6.) Run the program

```
IDL> test1
```

IDL
Data types

Data type	Size (bytes)	Variable Creation	Data Type Function
Byte	1	var = 0B	thisVar=Byte(var)
16-Bit Signed Integer	2	var = 0	thisVar=Fix(var)
32-Bit Signed Integer	4	var = 0L	thisVar=Long(var)
64-Bit Signed Integer	8	var = 0LL	thisVar=Long64(var)
Floating Point	4	var = 0.0	thisVar=Float(var)
Double Precision Floating	8	var = 0.0D	thisVar=Double(var)
String	0-32767	var = 'hi' or var = "hi"	thisVar=String(var)

IDL

Arrays

Data type	Size (bytes)	Array Name (all zeros)	Array Name (with values 0 → N-1)
32-Bit Signed Integer	4	INTARR	INDGEN
64-Bit Signed Integer	8	LON64ARR	L64INDGEN
Floating Point	4	FLTARR	FINDGEN
Double Precision	8	DBLARR	DINDGEN
String	0-32767	STRARR	SINDGEN

Arrays start with index 0!

IDL
General Remarks

Comments:

```
; this line is a comment
```

Exponentials:

```
a = b^2
```

Repeat until loop:

```
REPEAT BEGIN
[...]
ENDREP UNTIL (a EQ 10)
```

Do while loop:

```
WHILE (a LT 8) DO BEGIN
[...]
ENDWHILE
```

IF condition:

```
IF (a EQ b) THEN BEGIN
     print, 'a=b', a
ENDIF ELSE BEGIN
     print, 'a=/=b', a,b
ENDELSE
```

Do loop:

```
n = 8
a = findgen (n)
FOR i = 0, n-1 DO BEGIN
     print, i, a(i)
ENDFOR
```

Commands at the prompt

Line continuation:

```
IDL> result = alpha+beta* $
       (charlie*delta - echo)
```

Treated as one line

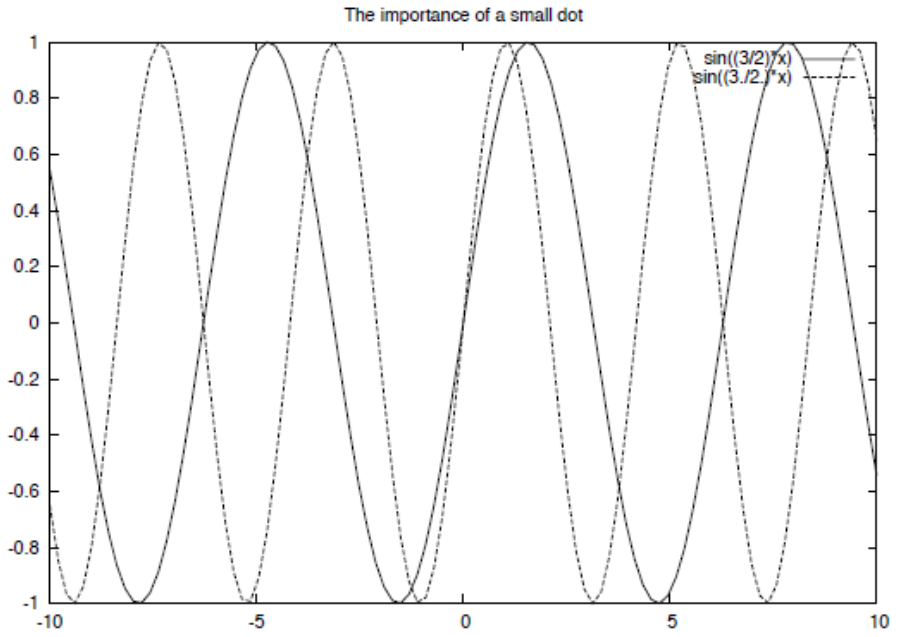
New lines that are part of a FOR LOOP

```
IDL> for i=0, n-1l do begin &
       print, i, data[i] &
       print, i, data2[i]
```

Treated as seperate lines in one FOR loop block

Data Types - In general

smalldot.ps

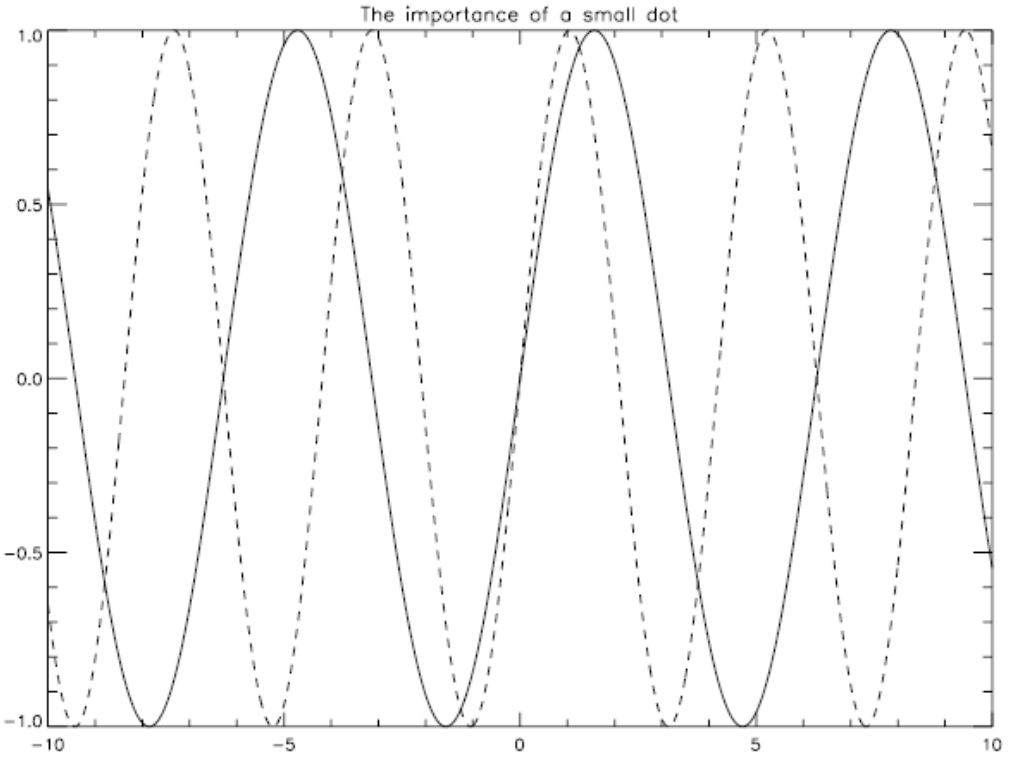


↑ Gnuplot $\sin[(3./2.) \cdot x]$

$\sin[(3/2) \cdot x]$

↓ IDL $\sin[(3./2.) \cdot x]$

$\sin[(3/2) \cdot x]$



idl_smalldot.pro, idl_smalldot.ps

Plotting routines

```
IDL> plot, x, y          ; x-y graph
IDL> oplot, x, 2.*y      ; x-y graph overplot
IDL> plots, x, y         ; x-y graph with symbols
```

Extra commands available for use with plot:

```
Thick=2      ; line thickness =2
color=red    ; line colour = red
psym=-4      ;  draw symbol type 4 connected with
               lines.
psym=4       ;  draw unconnected symbols
linestyle=2  ;  use dashed lines
```

Plot text strings 'P=(500,300)' at position 500, 300:

```
IDL> xyouts, 500, 300, 'P=(500,300)'
```

```
IDL> ?       ; online help
```

2D images

Simple routine for displaying 2D images – plots them in terms of pixels so small arrays may need interpolating up to fill the screen.

```
IDL> tvscl, dens                    ; x-y-graph
```

Imdisp displays images in a rescaled format – expands small arrays to fill the screen. Has more options such as data range selection.

```
IDL> imdisp, dens, range=[1.e-25,1.e-21]

IDL> contour, dens, x, y    ; contour plot

IDL> surface, dens                  ; generates a 3D surface
```

IDL

Create a postscript file:

```
IDL> olddev=!d.name
IDL> set_plot, 'ps'
IDL> device, filename = 'test1.ps'
IDL> x = findgen(1000)/100.
IDL> plot, x, sin(x)
IDL> device, /close_file
IDL> set_plot, olddev
```

IDL

Modules

Main program:

```
pro contour_go
    contour_start
    contour_main, plot_type = 1
end
```

IDL

Modules

Initialize the variables:

```
pro contour_start
common fileinfo, filename, fnr_start, fnr_stop, hsizearr
common coords, x, y, z, time
common vars, dens, temp

fnr_start = 1
fnr_stop  = 100
hsizearr  = 64
filename  = '/scratch/student1/NumPrak/test1.dat'

x = fltarr(hsizearr*2)
y = fltarr(hsizearr*2)
z = fltarr(hsizearr*2)
temp = fltarr(hsizearr*2, hsizearr*2, hsizearr*2)
dens = fltarr(hsizearr*2, hsizearr*2, hsizearr*2)
time = 0.

end
```

IDL

Modules

Main program:

```
pro contour_go
    contour_start
    contour_main, plot_type = 1
end
```

IDL

Modules

Create the plots:

```
pro contour_main, plot_nr = plot_nr
common fileinfo, filename, fnr_start, fnr_stop, hsizearr
common coords, x, y, z, time
common vars, dens, temp

for i = fnr_start, fnr_stop do begin
    contour_read, count = i
    contour, dens, x, y
endfor

end
```

IDL Virtual Machine

- Create compilable program (test.pro)
- Restore IDL-session:

```
.FULL_RESET_SESSION
```

- Compile your routine:

```
.COMPILE test.pro
```

- Include all used routines:

```
RESOLVE_ALL
```

- Create .sav file:

```
SAVE,/ROUTINES, FILENAME='/scratch/student1/NumPrak/test.sav'
```

- Run the program with IDL VM:

```
> idl -vm
```

IDL Program

Example

```
pro idl_read_file

!p.thick=2
!x.thick=2
!y.thick=2

labelsize = 1.5

temp = FLTARR(10000)
dens = FLTARR(10000)
pres = FLTARR(10000)

t = 0.
d = 0.
p = 0.

mH = 1.6726e-24
fracH = 0.75

count = 0L

filename = '/scratch/student1/NumPrak/rho_cheq.dat'
openr, lun, filename, /get_lun

print, lun

while (not EOF(lun)) do begin
    readf, lun, t, d, p
    temp(count) = t
    dens(count) = d * fracH / mH
    pres(count) = p
    count =  count + 1L
endwhile

temp = temp(0:count-1L)
dens = dens(0:count-1L)
pres = pres(0:count-1L)

olddev=!d.name
set_plot, 'ps'
device, filename='idl_rho_cheq.ps'

plot, temp, dens, xtitle='Temperature [K]', ytitle='Density [H cm!U-3!N]',$
      /ylog, /xlog, charsize=labelsize, $
      title='Thermal Equilibrium Function (!7q!3!Dcheq!N)'

device, /close_file
set_plot, olddev

end
```

Start and end of the program

pro <program name>

end

idl_read_file.pro

IDL Program

Example

```
pro idl_read_file

!p.thick=5
!x.thick=5
!y.thick=5
!p.charthick=5
labelsize = 2

temp = FLTARR(10000)
dens = FLTARR(10000)
pres = FLTARR(10000)

t = 0.
d = 0.
p = 0.

mH = 1.6726e-24
fracH = 0.75

count = 0L

filename = '/scratch/student1/NumPrak/rho_cheq.dat'
openr, lun, filename, /get_lun

print, lun

while (not EOF(lun)) do begin
    readf, lun, t, d, p
    temp(count) = t
    dens(count) = d * fracH / mH
    pres(count) = p
    count =  count + 1L
endwhile

temp = temp(0:count-1L)
dens = dens(0:count-1L)
pres = pres(0:count-1L)

olddev=!d.name
set_plot, 'ps'
device, filename='idl_rho_cheq.ps'

plot, temp, dens, xtitle='Temperature [K]', ytitle='Density [H cm!U-3!N]',$
      /ylog, /xlog, charsize=labelsize, $
      title='Thermal Equilibrium Function (!7q!3!Dcheq!N)'

device, /close_file
set_plot, olddev

end
```

System variables used here

```
!p.thick=2    ... thickness of the plotting line/ points
!x.thick=2    ... thickness of the x-axis
!y.thick=2    ... thickness of the y-axis
!p.charthick  ... character thickness
```

Others are for example:

```
!d.x_size  ... x-size of the window
!path      ... idl path (here idl finds your .pro files)
!pi        ... contains the constant pi
```

In general:

```
!p              ... main plotting variable
!p.background   ... background color
!p.charsize     ... character size
!p.font         ... character font
!p.position     ... position of the plot within the window
!p.multi[0,2,2,0,0]   ... for multiplots
        [0,*,*,*,*]   ... number of plots remaining
        [*,2,*,*,*]   ... number of plot columns per page
        [*,*,2,*,*]   ... number of plot rows per page
        [*,*,*,0,*]   ... number of plots in z-direction
        [*,*,*,*,0]   ... 0 for plots from left to right,
                          1 for plots from top to bottom
```

IDL Program
Example

```
pro idl_read_file

!p.thick=5
!x.thick=5
!y.thick=5
!p.charthick=5
labelsize = 2

temp = FLTARR(10000)
dens = FLTARR(10000)
pres = FLTARR(10000)

t = 0.
d = 0.
p = 0.

mH = 1.6726e-24
fracH = 0.75

count = 0L

filename = '/scratch/student1/NumPrak/rho_cheq.dat'
openr, lun, filename, /get_lun

print, lun

while (not EOF(lun)) do begin
    readf, lun, t, d, p
    temp(count) = t
    dens(count) = d * fracH / mH
    pres(count) = p
    count =  count + 1L
endwhile

temp = temp(0:count-1L)
dens = dens(0:count-1L)
pres = pres(0:count-1L)

olddev=!d.name
set_plot, 'ps'
device, filename='idl_rho_cheq.ps'

plot, temp, dens, xtitle='Temperature [K]', ytitle='Density [H cm!U-3!N]',$
      /ylog, /xlog, charsize=labelsize, $
      title='Thermal Equilibrium Function (!7q!3!Dcheq!N)'

device, /close_file
set_plot, olddev

end
```

Initialization of an array

```
temp = FLTARR(10000)
dens = FLTARR(10000)
pres = FLTARR(10000)
```

Others are for example:

```
BytArr ... Byte
IntArr ... 16-Bit Signed Integer Array
LonArr ... 32-Bit (Long) Signed Integer Array
DblArr ... Double Precision Floating Array
StrArr ... String Array
```

Also possible:

```
data = FLTARR(3, 10000)
[...]
data(0,count) = t
data(1,count) = d * fracH / mH
data(2,count) = p
[...]
data = data(*,0:count-1L)
plot, data(0,*), data(1,*), ...
```

IDL Program

Example

```
pro idl_read_file

 !p.thick=5
 !x.thick=5
 !y.thick=5
 !p.charthick=5
 labelsize = 2

 temp = FLTARR(10000)
 dens = FLTARR(10000)
 pres = FLTARR(10000)

 t = 0.
 d = 0.
 p = 0.

 mH = 1.6726e-24
 fracH = 0.75

 count = 0L

 filename = '/scratch/student1/NumPrak/rho_cheq.dat'
 openr, lun, filename, /get_lun

 print, lun

 while (not EOF(lun)) do begin
     readf, lun, t, d, p
     temp(count) = t
     dens(count) = d * fracH / mH
     pres(count) = p
     count =  count + 1L
 endwhile

 temp = temp(0:count-1L)
 dens = dens(0:count-1L)
 pres = pres(0:count-1L)

 olddev=!d.name
 set_plot, 'ps'
 device, filename='idl_rho_cheq.ps'

 plot, temp, dens, xtitle='Temperature [K]', ytitle='Density [H cm!U-3!N]',$
       /ylog, /xlog, charsize=labelsize, $
       title='Thermal Equilibrium Function (!7q!3!Dcheq!N)'

 device, /close_file
 set_plot, olddev

 end
```

Open a file to read it in IDL

```
openr, lun, filename, /get_lun

Attaches the file with a logical unit number:
          lun ... integer assigned to the data file
          /get_lun ... assignes an available int
```

Read the data from a file

```
Unfortunately not possible to write:
        readf, lun, t(count), d(count), p(count)
IDL Rule: You cannot read into a subscripted variable
```

If you know the number of rows:

```
for j=0, rows-1 do begin
  readf, lun, t, d, p
  [...]
endfor
```

openr & readf - for reading
openw & printf - for writing

IDL Program

Example

```
pro idl_read_file

 !p.thick=5
 !x.thick=5
 !y.thick=5
 !p.charthick=5
 labelsize = 2

 temp = FLTARR(10000)
 dens = FLTARR(10000)
 pres = FLTARR(10000)

 t = 0.
 d = 0.
 p = 0.

 mH = 1.6726e-24
 fracH = 0.75

 count = 0L

 filename = '/scratch/student1/NumPrak/rho_cheq.dat'
 openr, lun, filename, /get_lun

 print, lun

 while (not EOF(lun)) do begin
     readf, lun, t, d, p
     temp(count) = t
     dens(count) = d * fracH / mH
     pres(count) = p
     count =  count + 1L
 endwhile

 temp = temp(0:count-1L)
 dens = dens(0:count-1L)
 pres = pres(0:count-1L)

 olddev=!d.name
 set_plot, 'ps'
 device, filename='idl_rho_cheq.ps'

 plot, temp, dens, xtitle='Temperature [K]', ytitle='Density [H cm!U-3!N]',$
       /ylog, /xlog, charsize=labelsize, $
       title='Thermal Equilibrium Function (!7q!3!Dcheq!N)'

 device, /close_file
 set_plot, olddev

 end
```

Create a postscript file

```
olddev=!d.name
set_plot, 'ps'
device, filename='idl_rho_cheq.ps'

[...]

device, /close_file
set_plot, olddev
```

IDL Program

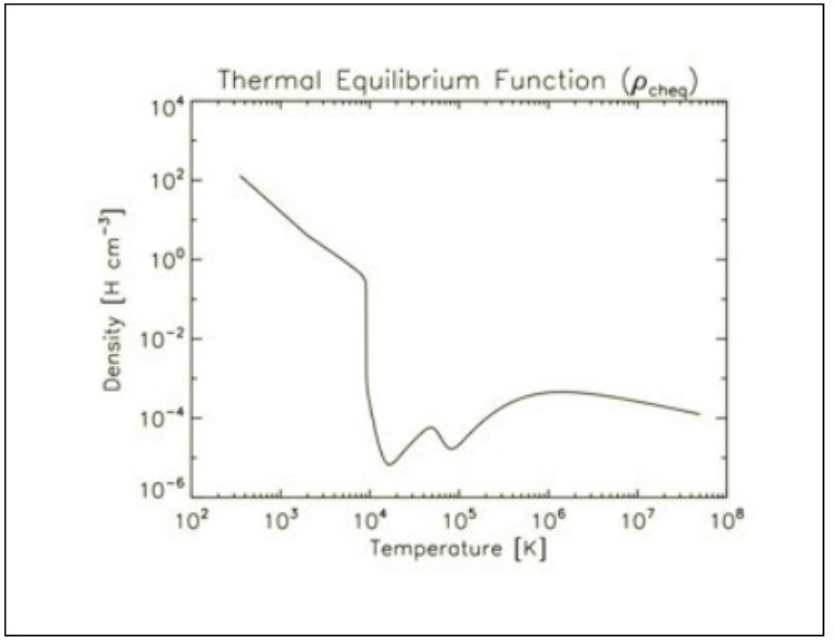
Example

```
pro idl_read_file

 !p.thick=5
 !x.thick=5
 !y.thick=5
 !p.charthick=5
 labelsize = 2

 temp = FLTARR(10000)
 dens = FLTARR(10000)
 pres = FLTARR(10000)

 t = 0.
 d = 0.
 p = 0.

 mH = 1.6726e-24
 fracH = 0.75

 count = 0L

 filename = '/scratch/student1/NumPrak/rho_cheq.dat'
 openr, lun, filename, /get_lun

 print, lun

 while (not EOF(lun)) do begin
     readf, lun, t, d, p
     temp(count) = t
     dens(count) = d * fracH / mH
     pres(count) = p
     count =  count + 1L
 endwhile

 temp = temp(0:count-1L)
 dens = dens(0:count-1L)
 pres = pres(0:count-1L)

 olddev=!d.name
 set_plot, 'ps'
 device, filename='idl_rho_cheq.ps'

 plot, temp, dens, xtitle='Temperature [K]', ytitle='Density [H cm!U-3!N]',$
       /ylog, /xlog, charsize=labelsize, $
       title='Thermal Equilibrium Function (!7q!3!Dcheq!N)'

 device, /close_file
 set_plot, olddev

 end
```

Plot

several options are possible
keywords for logarithmic plots: /xlog, /ylog

Subscript, superscript, greek letters!



IDL - gnuplot

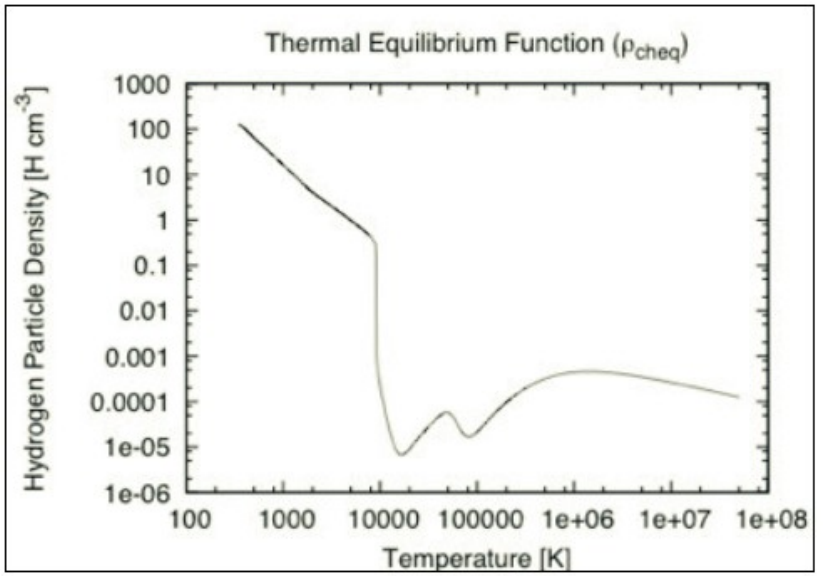
Syntax comparison

IDL

gnuplot

```
pro idl_read_file

 !p.thick=5
 !x.thick=5
 !y.thick=5
 !p.charthick=5
 labelsize = 2

 temp = FLTARR(10000)
 dens = FLTARR(10000)
 pres = FLTARR(10000)

 t = 0.
 d = 0.
 p = 0.

 mH = 1.6726e-24
 fracH = 0.75

 count = 0L

 filename = '/scratch/student1/NumPrak/rho_cheq.dat'
 openr, lun, filename, /get_lun

 print, lun

 while (not EOF(lun)) do begin
     readf, lun, t, d, p
     temp(count) = t
     dens(count) = d * fracH / mH
     pres(count) = p
     count =  count + 1L
 endwhile

 temp = temp(0:count-1L)
 dens = dens(0:count-1L)
 pres = pres(0:count-1L)

 olddev=!d.name
 set_plot, 'ps'
 device, filename='idl_rho_cheq.ps'

 plot, temp, dens, xtitle='Temperature [K]', ytitle='Density [H cm!U-3!N]',$
       /ylog, /xlog, charsize=labelsize, $
       title='Thermal Equilibrium Function (!7q!3!Dcheq!N)'

 device, /close_file
 set_plot, olddev

 end
```

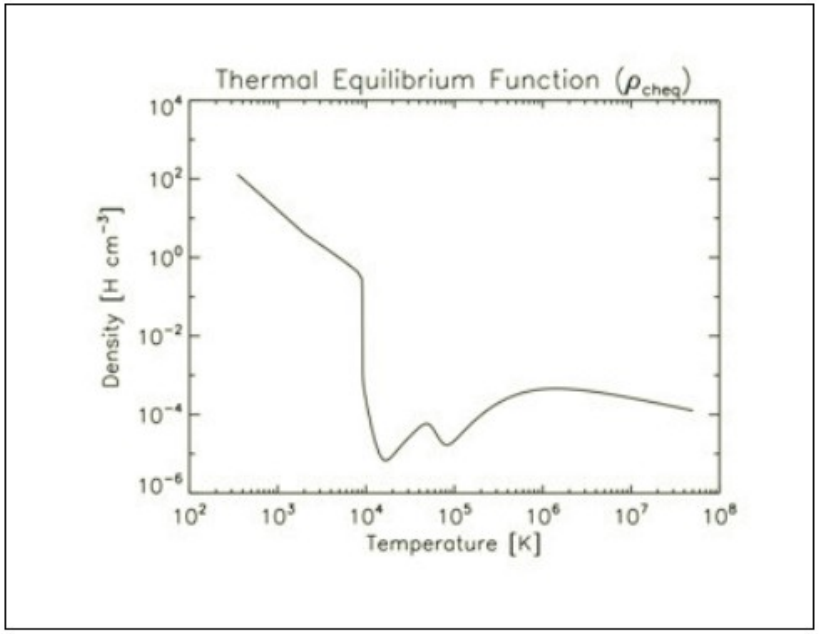
```
set size 0.6, 0.6
set xlabel "Temperature [K]"
set ylabel "Hydrogen Particle Density [H cm^{-3}]"
set title "Thermal Equilibrium Function ({/Symbol r}_{cheq})"
set logscale xy

mH = 1.6726e-24
fracH = 0.75

set term post eps enh
set output "gnu_rho_cheq.ps"

plot "rho_cheq.dat" u 1:($2*fracH/mH)  w l notitle

set term x11
```



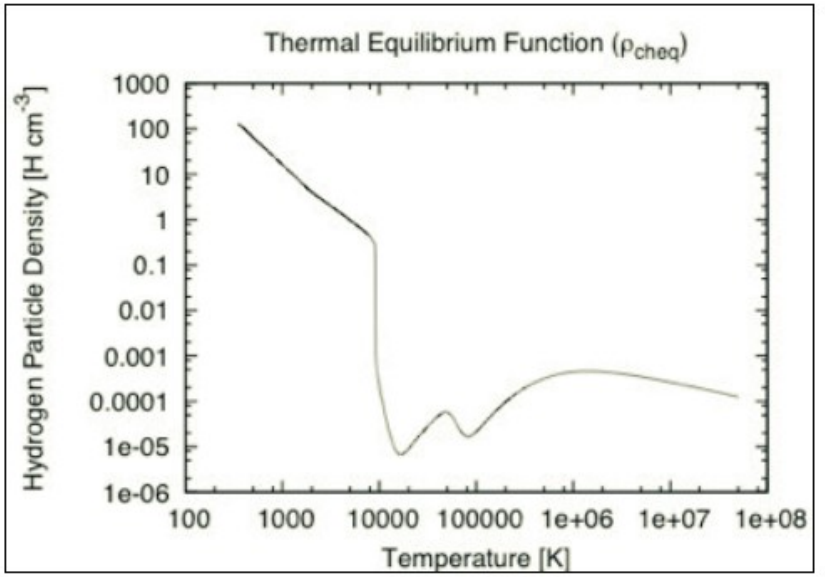
Gnuplot - IDL

Results

IDL



gnuplot



TexToIDL

- Addition of symbols and Greek letters in IDL awkward:

  !7q!3!Dgas!N → ρ_{gas}

- TexToIDL routines perform conversion of Latex formatted text into IDL's "language."

  > title1 = textoidl(" \rho_{gas} ")

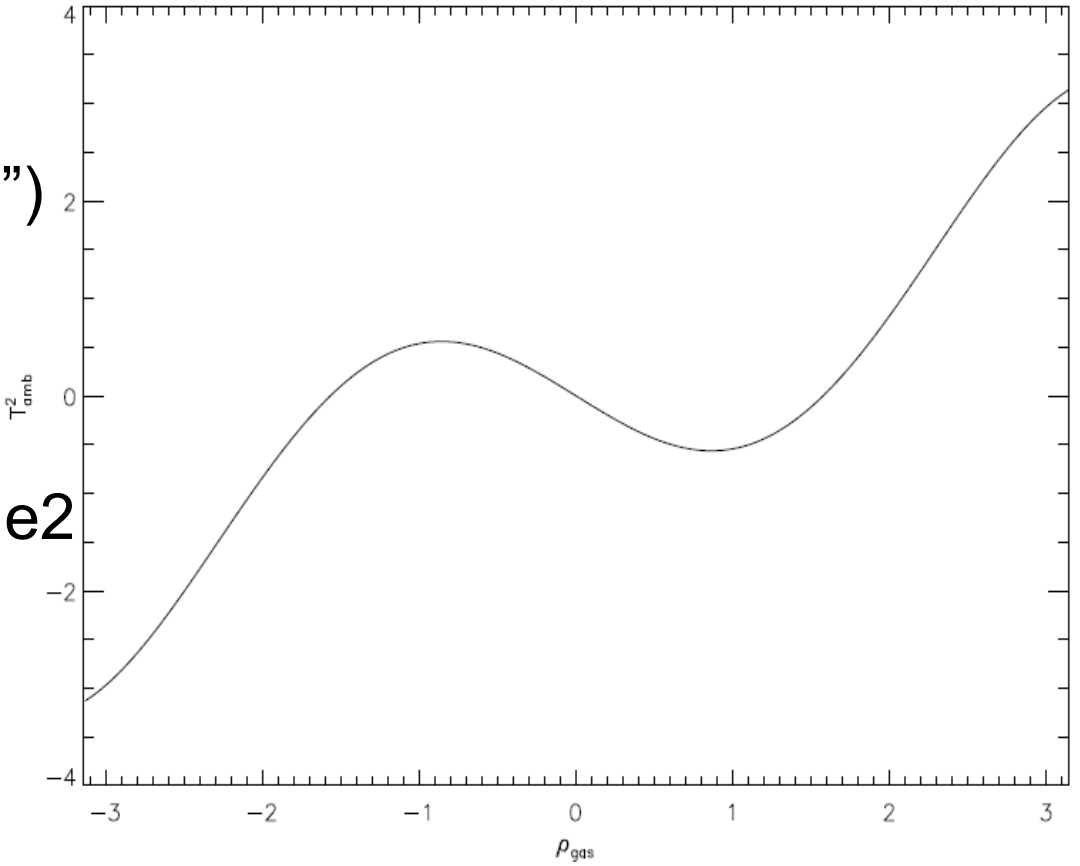
  > title2 = textoidl(" T^{2}_{amb} ")

  > print, title2

  T!S!U2!R!N!Damb!N

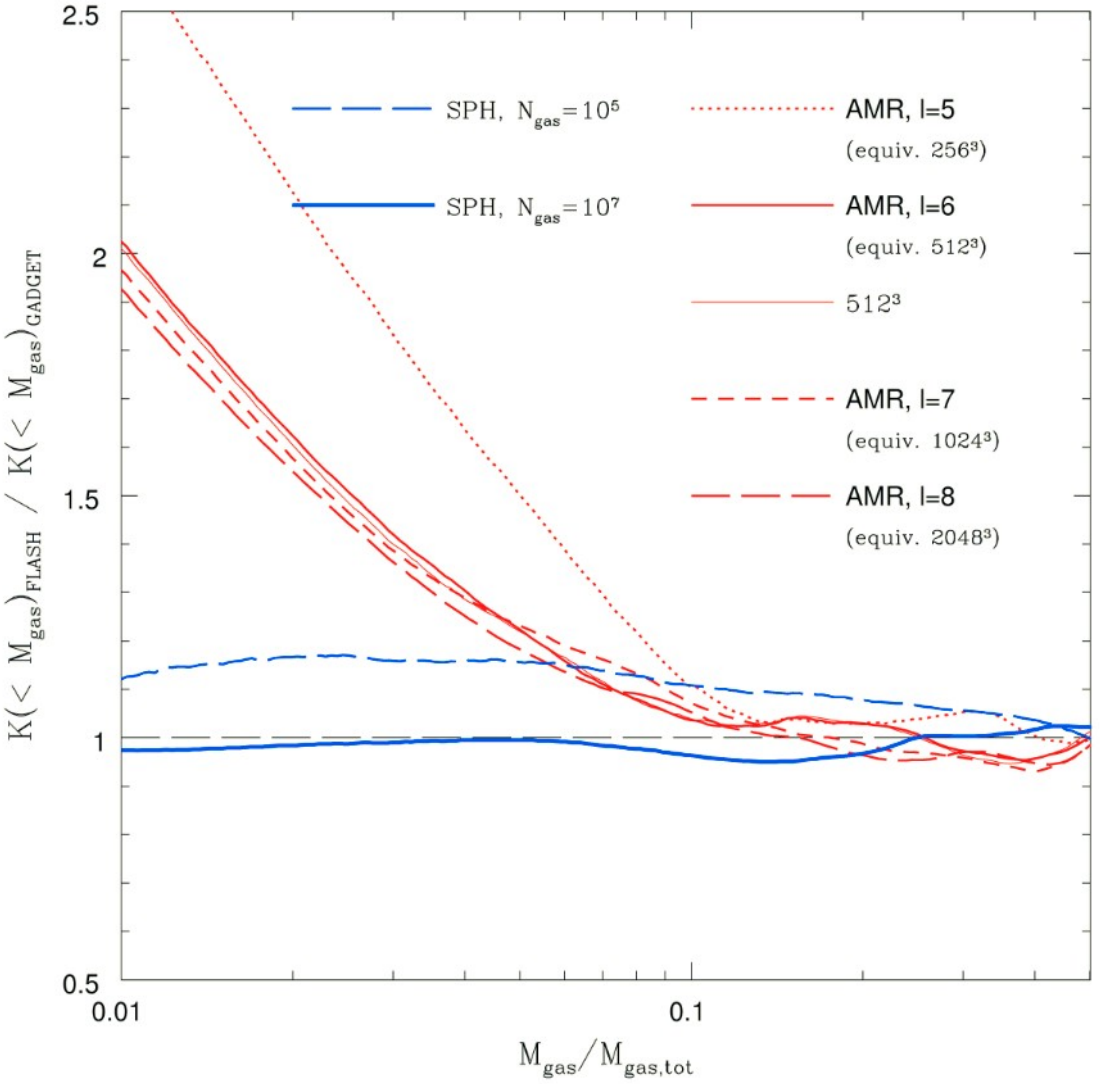
  > plot, x, y, xtitle=title1, ytitle=title2

- Gives the following



Legend commands allows you to add a legend to the plot

```
IDL> Legend, ['AMR i=5','AMR i=6','AMR i=7'], $
     color=[red,red,red],thick=[2,2,2], $
     Linestyle=[0,1,2], $
     /top, /right, box=0
```



Legend accepts most plot commands such as `psym` if you want to use symbols instead

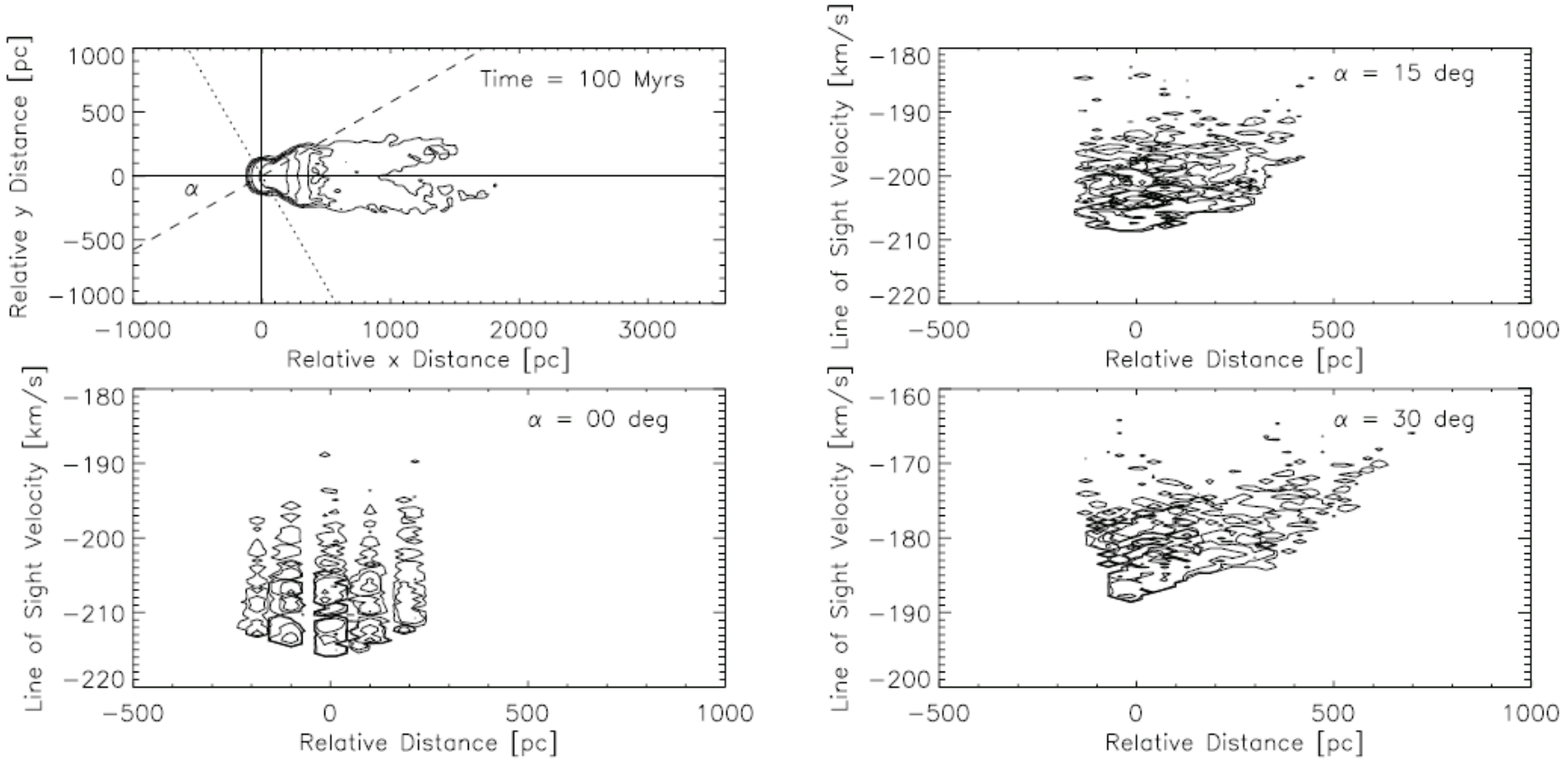
IDL Multiplot

Example

```
pro phasespace_main
[...]

contour, vardensc, x, y, position=[0.1,0.6,0.46,0.9]

for angle = 0, 30, 15 do begin
    phasespace_read, count = i
    phasespace_plot, angle = angle, count = i
    if (angle EQ 0) then begin
        cposition = [0.1,0.15,0.46,0.5]
    endif else begin
    if (angle EQ 15) then begin
        cposition = [0.59,0.6,0.95,0.9]
    endif else begin
    if (angle EQ 15) then begin
        cposition = [0.59,0.15,0.95,0.9]
    endif
    endelse
    endelse
    contour, dens, x, y, position=cposition
endfor
end
```

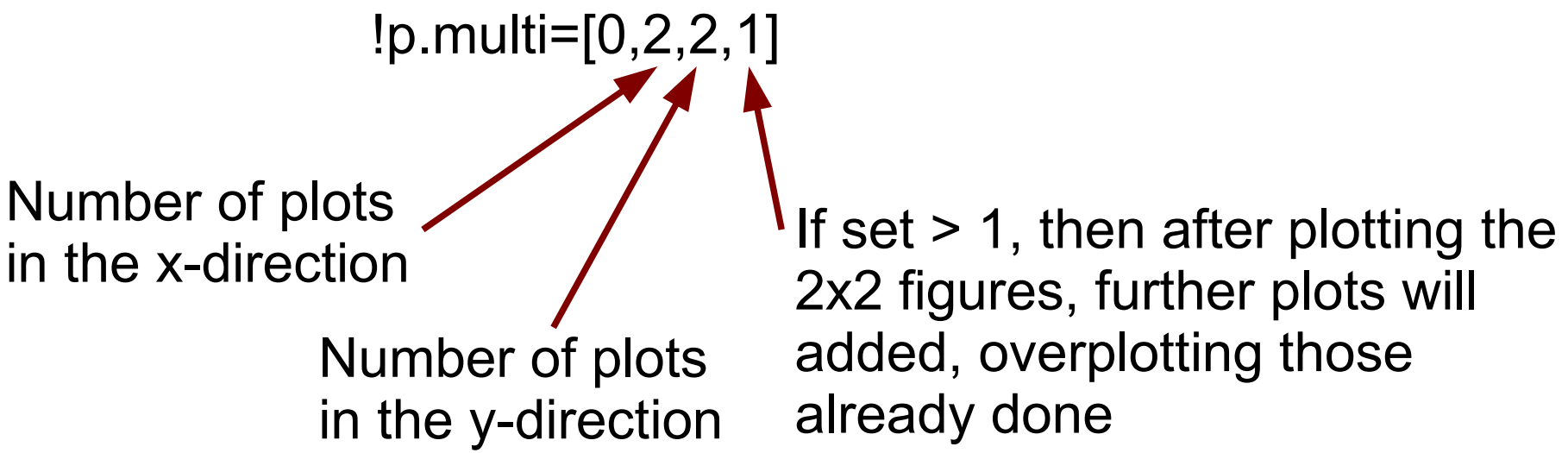
IDL Multiplot

Example



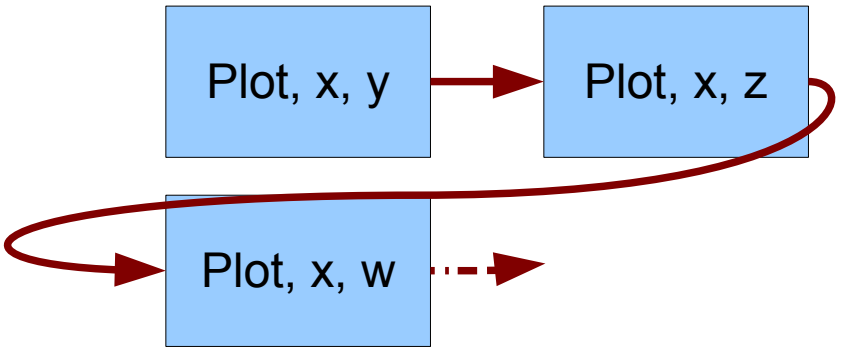
IDL Multiplot

- Or a quicker, although less versatile version:



- You then plot each figure as you would a conventional plot, e.g.

```
> plot, x, y
> plot, x, z
> plot, x, w
```



Presenting Data

- Think about what you want the plot to show
- Do not over-crowd a plot – too much data is difficult to follow
- You can:
  - represent scatter using error bars or shaded regions
  - Fit complex and noisey data using spline functions
- Postscript files record each point plotted → plotting millions of points will make them very large and slow to load
- Large numbers of data points can be reduced by:
  - Rebinning the data
  - IDL has an `nsum` command that only plots every `nsum` points
  - Randomly sample vast datasets
- Scientific journals often charge to print in colour so try and do less complicated plots in black and white

Example: Random sampling

```
> help, x
x       double  = array[1000000]
```

Want to plot a small random sample of these

```
> indexes = randomu(seed,1000)
```

Now have 1000 random values uniformly chosen between 0. and 1.

```
> indexes = indexes*(1000000.-1.)     ; normalise
> indexes_int = fix(indexes)
```

This will cause problems → idl normal integers only go up to 32768, should use long integers for numbers higher than this

```
> indexes_long = long(indexes)
> plot, x[indexes]
```

→ Plots 1000 randomly sampled points from x

```
> indexes = randomn(1000,seed)
```

Alternative that produces random points with a normal distribution

READCOL

```
> Readcol, 'file.dat', number, x, y, z,
f='(i,f,d,e)
```

- Reads in all lines of a formatted ascii file without needing to open/close file, or specify number of lines in the file
- Will skip all lines commented with a hash, #, or which do not fit the desired input format.
- User can specify format using the `f='()'` command or otherwise all values are assuming floating point
  - `i` = integer
  - `f` = float
  - `d` = double
  - `e` = scientific exponent notation (used for outputting data)

IDL Tips:

- If a function or subroutine goes wrong, IDL will stop where things go wrong:
  - Very usefull for debugging as it highlights the problem
  - Use `retall` to return IDL to its original state → this can help recover variables and arrays from routines that have crashed instead of re-running the entire program
- Try to perform array operations instead of loops as IDL is optimised for these
- IDL caches poorly → no memory cleanup whilst the program runs → programs will tend to slow down over time:
  - Try and modularise the problem so that you can perform sections at a time
  - Use `undefine,variable` to undefine memory before trying to overwrite it, e.g. Loading in the next file's density.

IDL Resources

- Download idl.tar from the numerical practical webpage
- In your home space (e.g. /scratch/student1) un-tar the ball of files using the command:

```
tar -xvf idl.tar
```

- It will create a folder called `IDL/pro/`
- Add the following to your `~/.bashrc` file modifying the necessary *student1* id and restart the shell:

```
export IDL_PATH=$IDL_PATH:+/scratch/student1/IDL/pro/

export TEXTOIDL_DIR=/scratch/student1/IDL/pro/TEXtoIDL/textoidl

export IDL_STARTUP=/scratch/student1/IDL/pro/idl_startup.pro
```

- In the file idl_startup.pro add any idl programs you want to automatically run when IDL starts
- READCOL, TexToIDL and other resources should now be automatically available in IDL for use

The GNU Debugger

Compile the program with the debug option “-g”

```
> gfortran -g program.F90 -o program.exe
```

(Note that code compiled with debug option lacks optimisation

→ will run slower)

Open manual using the command:

```
> Man gdb
```

Start the debugger:

```
> gdb program.F90
(gdb)
```

Common commands:

`(gdb) list`	lists the next few lines of the program.

Breakpoints – code stops each time it reaches these:

`(gdb) break 10`	Creates a breakpoint at line 10
`(gdb) break sub_test`	Creates a breakpoint at the start of subroutine "*sub_test.*"
`(gdb) info breakpoint`	Lists breakpoint numbers and positions
`(gdb) clear 1`	Clears breakpoint 1

`(gdb) run`	Runs the program
`(gdb) continue`	Continue running program
`(gdb) print var`	Prints the value of the variable or array *var*.
`(gdb) next`	Execute the current line and move forward
`(gdb) step`	Steps into the sub-routine at the current line
`(gdb) up`	Moves up a level in the subroutine hierarchy
`(gdb) down`	Moves down a level in the subroutine hierarchy
`(gdb) help`	Opens GDB documentation

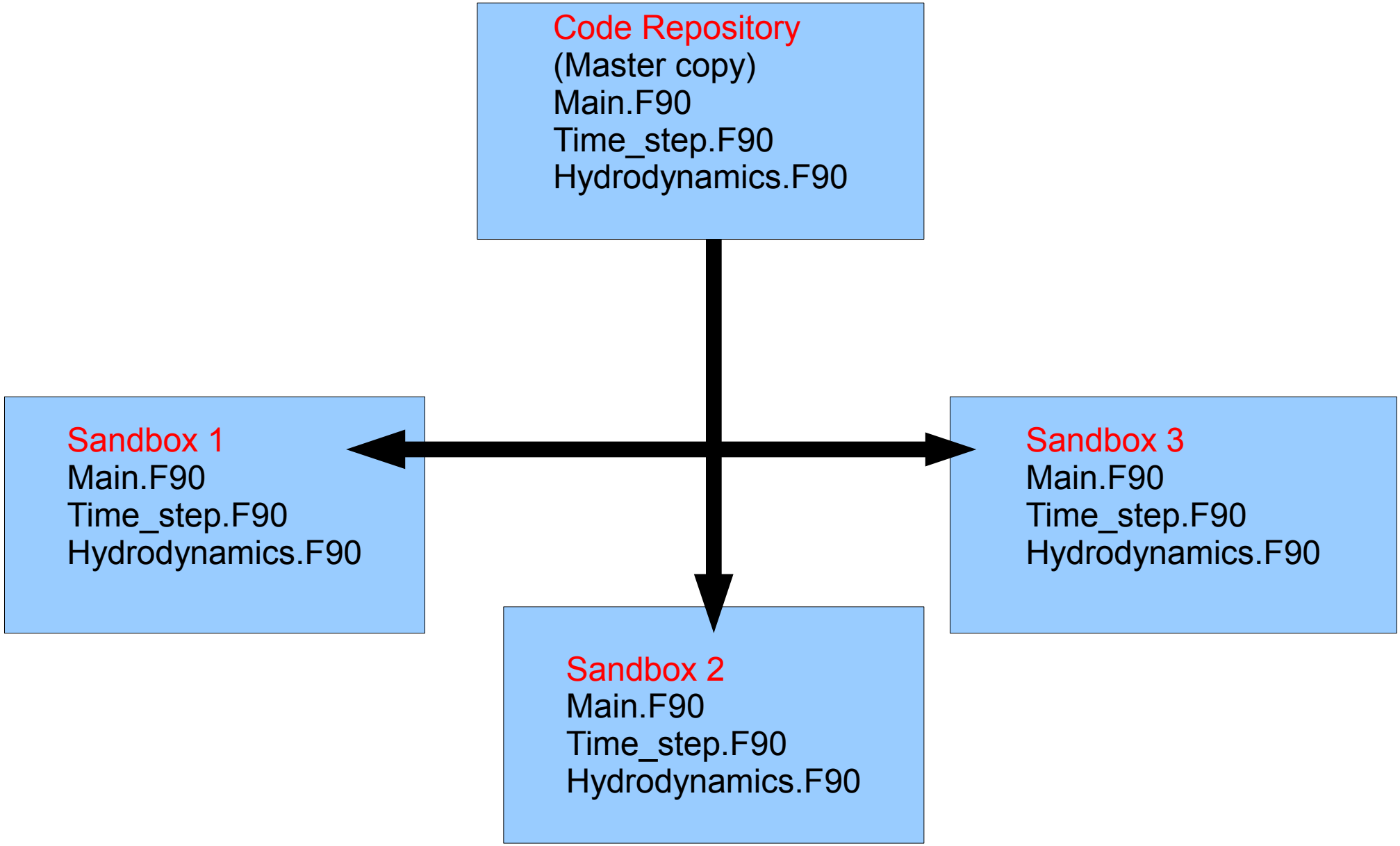
Source Version Management

- CVS – Current Version System
- SVN – Subversion Management

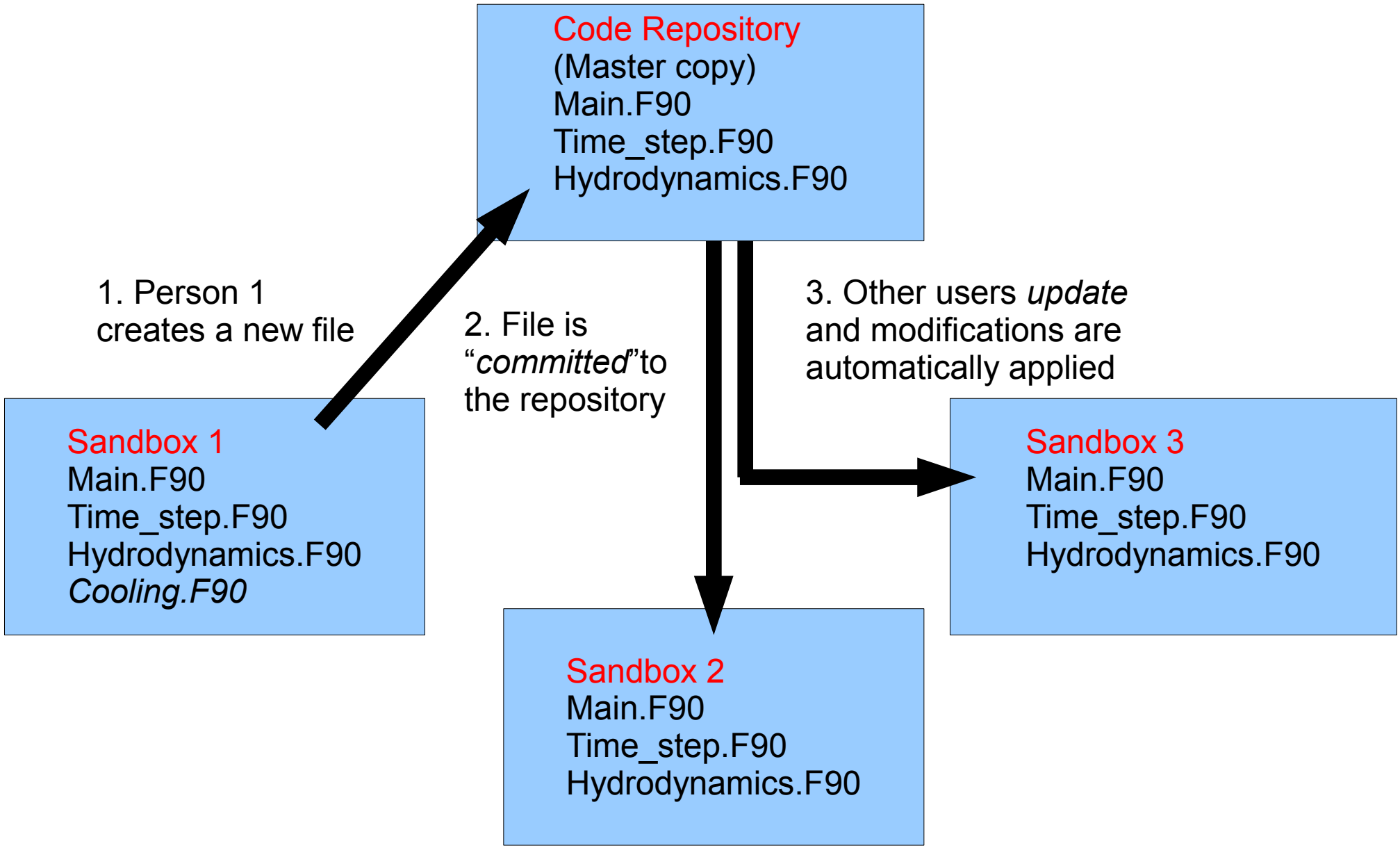
Purpose:

- Makes it easier for groups to collaborate on large software projects
- Can keep track of changes to the code
- Backs up files and can revert back to earlier versions if something breaks
- Provides basic checking for conflicts between users

How it works


Code Repository
(Master copy)
Main.F90
Time_step.F90
Hydrodynamics.F90
Sandbox 1
Main.F90
Time_step.F90
Hydrodynamics.F90
Sandbox 3
Main.F90
Time_step.F90
Hydrodynamics.F90
Sandbox 2
Main.F90
Time_step.F90
Hydrodynamics.F90

How it works


Code Repository
(Master copy)
Main.F90
Time_step.F90
Hydrodynamics.F90
1. Person 1
creates a new file
2. File is
"committed"to
the repository
3. Other users update
and modifications are
automatically applied
Sandbox 1
Main.F90
Time_step.F90
Hydrodynamics.F90
Cooling.F90
Sandbox 2
Main.F90
Time_step.F90
Hydrodynamics.F90
Sandbox 3
Main.F90
Time_step.F90
Hydrodynamics.F90

CVS

- Create repository (Master copy of the code)

```
> mkdir $HOME/cvsroot

> cvs -d $HOME/cvsroot init
```

- Create project

```
> cd /scratch/student1/NumPrak/hydro/

> cvs import -m "start version" hydro vendor start
```

- Set system variables (e.g. In ~/.bashrc)

```
export CVSROOT=$HOME/cvsroot/

export CVSROOT=:ext:student1@lab1...:cvsroot/
```

- In sandbox (local copy of the code we change):

```
> mkdir $HOME/sandbox; cd $HOME/sandbox

> cvs checkout hydro
```

→ copies files from CVS project "hydro" into the current directory

- Update sandbox version

```
> cvs update
```

- Check to see which files you have changed recently

```
> cvs status
```

- Check differences between local and repository files

```
> cvs diff main.F90
```

- Commit modified files to the CVS master copy in the repository:

```
> cvs commit main.F90 -m "comments on what was
                          changed"
```

- Print to screen a list of all modifications made with times and which user made them

```
> cvs log
```

- Revert back to an older versions (#12)

```
> cvs revert 12 main.F90
```

Conflicts

- If two people work on the same routine then their modifications may conflict
- Users are forced to update their files to the most recent version of the code before they can submit their modifications

```
> cvs update
```

- Areas of code which do not conflict are merged

```
> cvs update

> retrieving revision 1.3

> Merging differences between 1.2 and 1.3 into
  hydro.F90
```

- Obvious changes to the same lines of code are highlighted as conflicts → requires user intervention

```
> Rcsmerge: warning: conflicts during merging

> cvs update: conflicts found in hydro.F90
```

Conflicts

- Conflicts are highlighted in a pre-merged file, *hydro.F90.1.3*

```
>>>>>>>>>>>
do i=0, n-1
   dens[i]=dt*flux[i-1]
enddo
==============
do i=0, n-1
   dens[i]=dt*flux[i-1]
enddo
<<<<<<<<<<<
```

- Manually resolve conflicts and commit the new file

```
> cvs commit
```