VMD hands on:
Structural file manipulation

Rogelio Rodríguez-Sotres,
Bioquímica, FQ, UNAM.
2022

There are two ways of interacting with VMD

- Graphical widgets: clicking and interacting with graphical assistants
- The TK console: A programmable interface to an interpreted language called tcl/tk.
- If installed and started with python support, the interface has a python/tk-inter interpreter (python 2 only).

In this tutorial series:

The use of tcl/tk commands in VMD will be revised

SOME NOTES ON THIS PRESENTATION:

This is a the slide title

THIS IS A GENERAL COMMENT

% THIS IS A COMMAND IN VMD-TK-CONSOLE

$ THIS IS A COMMAND IN A BASH TERMINAL

> THIS IS ALSO A COMMAND IN A TERMINAL

This is a secondary comment

% this is a command in a tk-console... again

$ this is a command in a terminal, yet again

> this is a command in a terminal, once more

- this is a note, or the computers response

sizes and colors change emphasis, NOT meaning

IMPORTANT SYMBOLS IN THE PRESENTATION

USE OF SOME KEY'S IS SHOW AS FOLLOWS:

$ ⏎ or ENTR means press enter

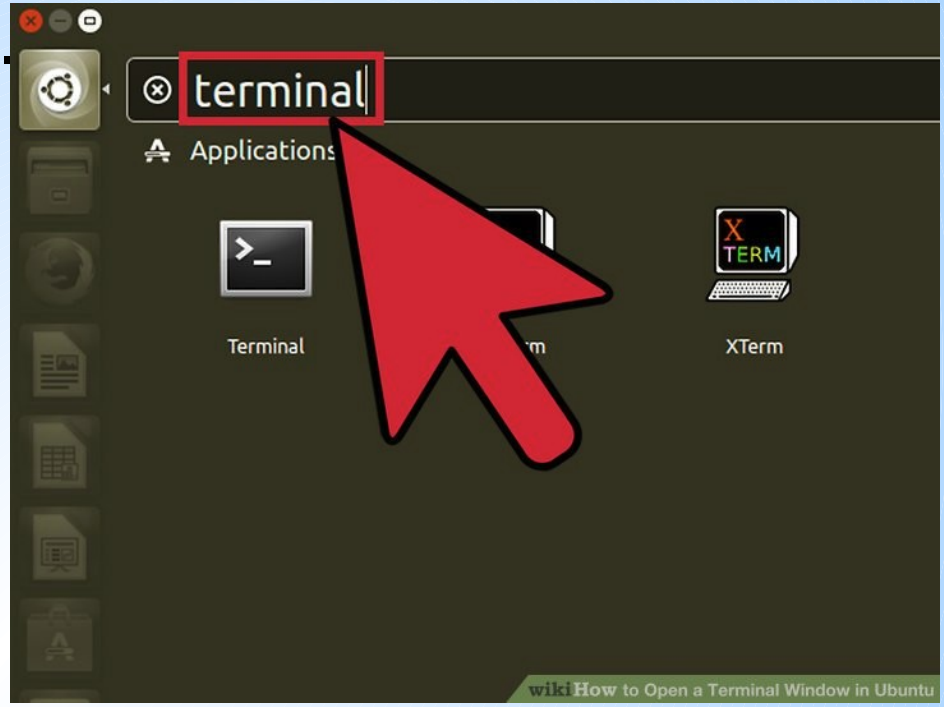
ESC press ESCAPE key

$ ⇧ CTRL ⌥ are composed key usually press with some other, for example: CTRLⒸ means presing control an C keys toghether

←↑↓→ press left, up, down & right arrow key, respectively

FN① indicates pressing the funtion 1 key

Let us start. Open a terminal .



> cd $HOME⏎

> mkdir session2⏎

> cd session2⏎

Now, we need some file to process...

Now we are on a folder named session2

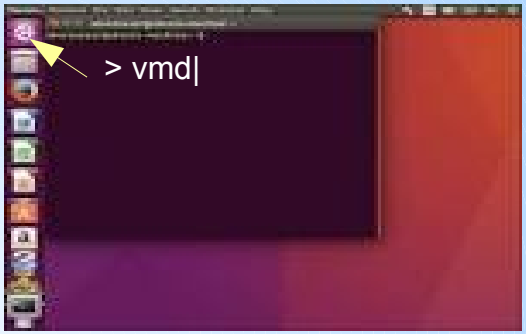
Let us proceed to download some pdb files

from:

on windows, simply open VMD and go to

extensions => Tk console

depa.fquim.unam.mx/proteinas/mdcb/model/data/PDBfiles4examples.zip



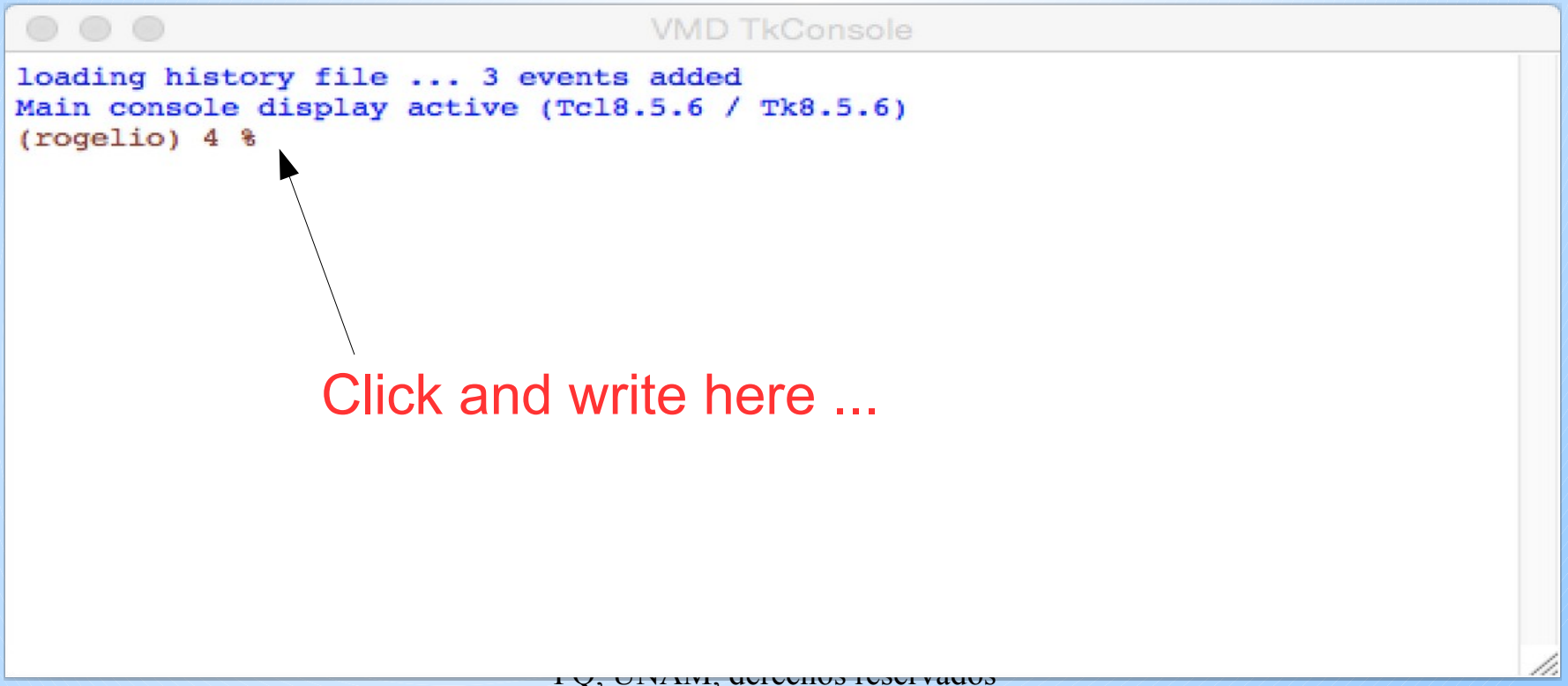
Open a new terminal

> cd $HOME/session2⏎

> vmd⏎

> vmd> menu tkcon on⏎

the tk/console appears...



ON TCL/TK

tcl/tk is an interpreted computer language

- # documentation at: www.tcl.tk/doc/
- # It is a POSITIONAL language, meaning:
- # commands (CMD) are followed by arguments (ARG)
- # **Order is strict**, if input is WRONG, so will be the outcome
- # It has a quick built-in HELP

> mol↵

- usage: mol <command> [args...]
- Molecules and Data: ...
- ....
- See also

> molinfo↵

#More on VMD-tcl/tk

Tcl/Tk **in VMD** has:

"general commands" & "mol commands"

```
# examples of general commands
% expr "2+2^2"⏎
```

→ makes the calculation

- 6

```
% set yo "Rogelio"⏎
```

→ creates "yo" and fills it

- Rogelio

```
% puts "hola mundo, me llamo $yo"⏎
```

→ tcl answer

- hola mundo, me llamo Rogelio → What?

```
% puts {hola mundo, me llamo $yo}⏎
```

- hola mundo, me llamo $yo → What?

```
# EXPLANATION:
# Strings in quotes are scanned and substituted, before passing to the command
# Variables like "yo" contain data and $yo points to its content:  "yo⇒Rogelio"
# the command set  takes a variable name, 1st arg and fils it with content
```

summary

commands in tcl have the form:

% CMD arg❶ arg❷ arg❸ …↵

the next CMD is not equivalent, because or ARG order

% CMD arg❸ arg❶ arg❷ …↵

NOTE: some comands may accept flags, example:

% puts -nonewline "Hola mundo"↵

Usualy, the flag is optional

Variables

Any name can be a variable, except for "reserved words"

Reserved words are those predefined: puts, set, etc…

their content can be a string or a list. Numbers are strings

list elements can be strings or a list, so lists can be nested

variable can be arrays with one, two or more indexes:

var(0), var (1), etc.

var(0,0), var(0,1)

indexes can be alfanumeric

var(a), var(b), etc

examples

```
% set me “free”                # me contains “free” as string
% set letras { ax by cz } # letras constains a list of strings
% puts $letras
> ax by cz
% lindex $letras 2    # retrieves the 3rth element of the list
> cz
% lappend letras {$letras} # adds to string $letras at “letras” end
% puts $letras
> ax by cz $letras
% lappend letras $letras #duplicates “letras” content
> ax by cz $letras ax by cz $letras
% foreach itm $letras { puts $itm} # runs over the list and prints
> ax
> by
  ...
```

more examples

```
% set nums { 12 + 16 + 28 +  45} # nums is a list strings
% expr $nums
> 101     # now explain the result!
% set arry(0) 25
% set arry(1) 100
% puts $arry
> can't read "arry": variable is array
# Arrays are indexed blocks of strings, but the index does not have to
  be numeric.
% set arry(a) “abba”
% puts “ARRY: $arry(0), $arry(1), $arry(a)”
> ARRY: 25, 100, abba
% foreach itm $arry {puts $itm}
> can't read "arry": variable is array  # but lists cannot be run like
  lists
```

More on VMD-tcl/tk

Commands can be nested using [ ]

```
% puts "hola mundo, son las [clock format [clock seconds] -format
  {%H:%M}]"⏎
```

contesta la hora del día en s

escribe

da formato a la hora

¿y esto?

```
# EXPLANATION:
# (1) The hour is determined in ms, (2) then replaces the inner [cmd ⓐ ⓑ ⓒ…]
# (3) It is formated according to the optional flag -format {%H:%M}, meaning  hh:mm
      # The result replaces the outer [cmd ⓐ ⓑ ⓒ…] and
(4) then the line is executes, i.e. writes a string
```

- hola mundo, son las 17:20

NOW LET'S GO BANANAS (monkey business)

make sure you are in the right folder

% pwd # (same CMD as in bash)

- /home/sica/session2

If you are out of place, move into folder where your pdb files are:

% cd “~/Documentos/session2”

% mol new "⊡⊡⊡⊡.pdb"⏎ # we loaded a file in memory

- 0

(here ⊡⊡⊡⊡ is the code of a pdb file of your choice)

"0" is the "molid" *i.e.* a number given to your molecule

% mol new "⊡⊡⊡⊡.pdb"⏎ # a second molecule is loaded

% mol new "⊡⊡⊡⊡.pdb"⏎ # a third molecule is loaded

molinfo list⏎ # info about the molecule

- 0 1 2

% mol delete 1

% molinfo list # ¿What’s going on?—explain the result

The active molecule

```
#   Only one molecule has active focus at any time
%   molinfo top⏎
    ▸ 2

#   "molinfo top" tell us the molid of the ACTIVE molecule
#   many command act on this molecule by default
#   you can type top insted of its number.
#   we can change the ACTIVE molecule with
#   mol top N⏎ # N is any mol-ID integer
%   mol top [lindex [ molinfo list ] 1 ]⏎
```

returns a list of loaded molecules

reads a list and returns element 0

changes the ACTIVE molecule

Explanation
The most inner [] is replaced with a list ofIDs of loaded molecules
lindex examines the list, and returns the element Nº 2 (the second)
"mol top" makes "ACTIVE" that particular molecule

¿Qué pasa con el comando “mol top 1”?

more loading

```
#  VMD number molecules without reusing numbers
#  in the graphics window, only two loaded "mols" are shown
%  mol top 0 # la molécula activa es ahora la cero
%  mol addfile "⊡⊡⊡⊡.pdb"⏎ #añadimos un pdb sobre si mismo
%  molinfo top get numframes
#  ahora VMD tiene el doble de copias de la molécula dentro del mismo
   espacio de memoria.
#
```

ATOMSELECT COMMAND!!

atomselect creates a command with subcommands

- # these act on atoms chosen by the second argument (string)
- # the molid used for the selection is the first argument

% set allprot [atomselect top "protein"]⏎

which atoms?

which molid?

where the result is stored

% set allelse [atomselect top "not protein"]⏎

- # The *set* command is used to retain the name of the new CMD
- # "$allprot" and "$allelse" will call those commands

Now *$allprot* is a command that acts on the protein,
while *$allelse* acts on the "non-proteianeceous" atoms

ATOMSELECT COMMAND!!

$allprot and $allelse commands have subcommands↵

% $allprot get {beta} # lists B factors for "protein" atoms

% $allprot set {beta} 1.0↵ # set protein B factors to 1.0

% $allelse set {beta} 3.0↵ # set other B factors to 3.0

% set mybb [atomselect top "(name H N CA CO O OXT) and (protein)"]↵

% $mybb get {resid name}↵

VMD lists the BB atoms by residue number and type

as a list of lists (each element is a list of properties).

- { {ele00 ele01 } { ele10 ele 11} ... }

#We can now save atoms on a selection

```
%$mybb writepdb "truebb.pdb"
  % $mybb set beta "8.0"⏎
  % $mybb writepdb "betamod.pdb"⏎
  # we have store two BB atom sets with different B-factors
  # Let us see the result
  % mol selection "all"⏎
  % mol representation licorice 0.3 90 90⏎
  % mol color beta⏎
  % mol material "EdgyShiny"⏎
  % mol delrep top 0⏎
  % mol addrep top⏎
  # see the result in the OpenGL display
```

this acts on TOP and is for graphics

how are atoms drawn?

how to paint them?

what texture to apply?

delete representation &

add a new representation

#We can now save atoms on a selection

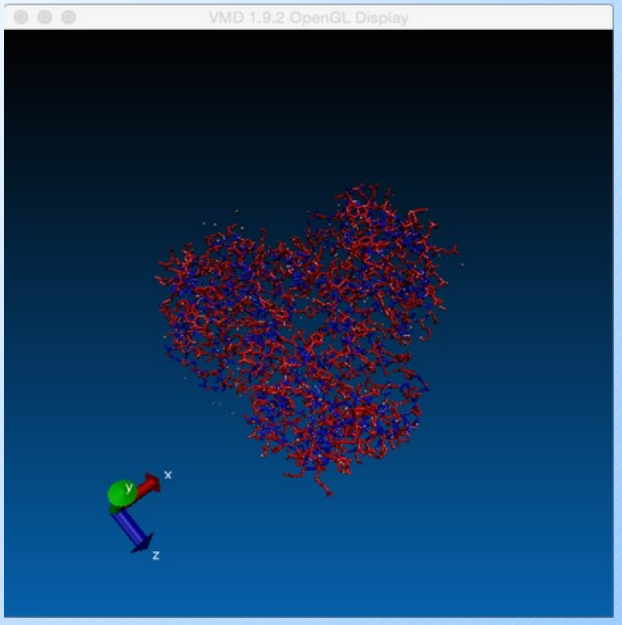
$mybb is a command with subcommands, acting on selected atoms:

```
% $mybb⏎

# usage: <atomselection> <command> [args...]
# Commands for manipulating atomselection metadata:
#   frame [new frame value]      -- get/set frame
#   molid|molindex       -- get selection's molecule id
#   text                 -- get selection's text
#   delete               -- delete atomselection (to free memory)
#   global               -- move atomselection to global scope
#   update               -- recalculate selection
# Commands for getting/setting attributes:
#   num                  -- number of atoms
#   list                 -- get atom indices
#   get|get <list of attributes>      -- same as 'atomselect keywords'
#   getbonds | setbonds <bondlists>    -- get (list) or set bonded atoms
#   getbondorders | setbondsorders <bondlists> -- get or set list of bond orders
#   getbondtypes|setbondtypes   <bondlists>  -- get or set list of bond types
#   moveto|moveby <3 vector>      -- change atomic coordinates
#   lmoveto|lmoveby <x> <y> <z>
#   move <4x4 transforamtion matrix>
# Commands for writing to a file:
#   writepdb <filename>     -- write sel to PDB file
#   writeXXX <filename>  -- write sel to XXX file (if XXX is a known format)
```

change the environment of the representation

```
% display backgroundgradient 1⏎
% display resetview⏎
# the changes allowed us to color the sidechains, backbone
# and heteroatoms with different colors
# we can also encode information in these fields
# In addition to PDB fields, VMD has three user fields
# USER1, USER2 and USER3
```



Let us do aggressive changes to molecular information

```
% $mybb get {resname}⏎
% $mybb set {resname} {GLY}⏎
# Now all backbone atoms are labelled as GLY, i.e.
# sequence information is lost for these atoms
% $mypdb writepdb "naked_John_Doe.pdb"⏎
# the resulting PDB file is a naked BB with no identity
% quit⏎
> gedit "naked_John_Doe.pdb"
```

How can you find those things that have changed?

VMD commands can be in a file and called as scripts

```
#  Find out where is vmd program and edit script
>  which vmd⏎ # usually /usr/local/bin/vmd
>  gedit pdb2polyAA⏎
```

????? write here the line that starts a BASH script!!

```
#Comment to hide BASH call, it must end with \↵
exec /usr/local/bin/vmd -dispdev text -e "$0" -args ${1+"$@"}↵

if { [llength $argv] < 3 } { puts "error: missing arguments"; quit}↵
     mol new↵
     mol addf le [lindex $argv 0]↵
 #   argv stores the arguments added at the command line
 #   we shall need three arguments 0:input.pdb 1:AAA 2:output.pdb
set protbb [atomselect top "(type H N CA C CO O OXT ) and ( protein )" ]↵
set newaa  [lindex $argv 1]↵
$protbb set {resname} $newaa↵
$protbb writepdb [lindex $argv 2]↵
quit↵
 #   SAVE and exit
```

let us run it

give it execution permission

$ chmod 755 pdb2polyAA

$ ./pdb2polyAA myfile.pdb ALA allala.pdb

Now you can make several instances of the pdb_bb
with different monotonous aa sequences (all wrong)

¿Can we fix the protein data?

Using Rosetta design in Fixed backbone mode

- Roseta design module is named fixbb.xxxxxxx
  - xxxxxx is the compilation form usually linuxgccrelease
- to called we can use a flags file or give options in the command line
- in barracuda you need to load the module "rosetta/??"
- in other systems you need to set up the environment as requested in "rosetta instalation instructions"
- ROSETTA3_DB environment variable sould be set and point to rosetta database forlder in the rosetta instalation folder.

Setting up resfile input

we shall need a rosetta "resfile" to indicate how fixbb is going to redesign the structure.

- In this case we need to instruct the software to rebuild all the positions with complete freedom of choice:
- i.e. put any compatible aminoacid residue at each one of the positions of the protein chain. The resfile should look like this:

```
ALLAA

start
```

- this is a very simple instructions file. ALLAA means that any AA choise is fine (as long as it fits well into the protein backbone).

Set up flags file

the minimal "flags" file should be:

```
-l pdb4rbld.lst
-resfile r3_4XTB_frkA.res
-nstruct 5
```

this meand do the reconstruction 5 times

Here we need to list all pdb files to rebuild in the file pdb4rbld, this can be quickly done using:

```
$ ls -1 *_frk???.pdb > pdb4rbld.lst
```

here we assume the files were named as xxxx_frkAaa.pdb, xxxx_frkAab.pdb and so on

running **Rosetta** is now simple

> fixbb.xxxxxxxx @flags > run.log **2>&1** &⏎

when the run is finished you should find many new pdbfiles ending in:

_0001.pdb _0002.pdb ... etc.

Now what should we do with these?

you could see them in vmd with:

% for each npdb [glob "*.pdb"] {mol new $npdb} ⏎

OJO: "**2>&1**" IS ONE WORD, with NO SPACES in it.

Hmmer Search

extract all fasta sequences form pdb files

$ pdb2fasta *_0???.pdb > xxxx_Xrd.seq⏎

Next create a hmmer estatistical device

$ hmmbuild xxxx_Xrd.hmm xxxx_Xrd.seq⏎

Search the sequence database:

$ hmmsearch xxxx_Xrd.hmm /home/dbr/uniprot_sprot.fasta > xxxx_Xrd.srch_uniprot 2>&1 &↵

$ Now we need to see the results in xxxx_Xrd.srch_uniprot

Results

```
# hmmsearch :: search profile(s) against a sequence database
# HMMER 3.1b1 (May 2013); http://hmmer.org/
# Copyright (C) 2013 Howard Hughes Medical Institute.
# Freely distributed under the GNU General Public License (GPLv3).
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# query HMM file:                  AtPPa1_AF_a-0.hmm
# target sequence database:        /home/dbr/uniprot_sprot.fasta
# max ASCII text line length:      unlimited
# sequence reporting threshold:    E-value <= 10
# sequence search space set to:    10000000
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Query:       AtPPa1_AF_a-0  [M=212]
Scores for complete sequences (score includes all domains):
   --- full sequence ---   --- best 1 domain ---    -#dom-
    E-value  score  bias    E-value  score  bias    exp  N  Sequence              Description
    ------- ------ -----    ------- ------ -----   ---- --  --------              -----------
    2.2e-41  150.8   0.0    2.3e-41  150.7   0.0    1.0  1  sp|Q93V56|IPYR1_ARATH  Soluble inorganic pyrophosphatase 1 OS=Arabidopsis thaliana OX=3702 GN=PPA1 PE=1 SV=1
    3.5e-41  150.1   0.1    3.9e-41  149.9   0.1    1.0  1  sp|O82597|IPYR5_ARATH  Soluble inorganic pyrophosphatase 5 OS=Arabidopsis thaliana OX=3702 GN=PPA5 PE=2 SV=1
    3.6e-40  146.8   0.0    4.1e-40  146.6   0.0    1.0  1  sp|O48556|IPYR_MAIZE   Soluble inorganic pyrophosphatase OS=Zea mays OX=4577 GN=IPP PE=2 SV=1
    4.3e-40  146.5   0.0    4.8e-40  146.4   0.0    1.0  1  sp|O82793|IPYR3_ARATH  Soluble inorganic pyrophosphatase 3 OS=Arabidopsis thaliana OX=3702 GN=PPA3 PE=2 SV=1
    1.6e-39  144.7   0.0    1.8e-39  144.5   0.0    1.0  1  sp|Q43187|IPYR_SOLTU   Soluble inorganic pyrophosphatase PPA1 OS=Solanum tuberosum OX=4113 GN=PPA1 PE=1 SV=1
    2.4e-39  144.1   0.0    2.7e-39  143.9   0.0    1.0  1  sp|A2X8Q3|IPYR_ORYSI   Soluble inorganic pyrophosphatase OS=Oryza sativa subsp. indica OX=39946 GN=IPP PE=2 SV=1
    2.4e-39  144.1   0.0    2.7e-39  143.9   0.0    1.0  1  sp|Q0DYB1|IPYR_ORYSJ   Soluble inorganic pyrophosphatase OS=Oryza sativa subsp. japonica OX=39947 GN=IPP PE=2 SV=1
    1.5e-38  141.5   0.0    1.7e-38  141.3   0.0    1.0  1  sp|Q9LFF9|IPYR4_ARATH  Soluble inorganic pyrophosphatase 4 OS=Arabidopsis thaliana OX=3702 GN=PPA4 PE=1 SV=1
```

Headings, HMM model name (query), target sequences, etc

List of meaningfull hits (above threshold)

this is different only if your match is discontinuos

bias should be small

Score is a ratio of Log(Probability HMMmodel)/Log(Probability random model)

The smaller the E-value, the higher statistical significance

```
Domain annotation for each sequence (and alignments):
>> sp|Q93V56|IPYR1_ARATH  Soluble inorganic pyrophosphatase 1 OS=Arabidopsis thaliana OX=3702 GN=PPA1 PE=1 SV=1
   #    score  bias  c-Evalue  i-Evalue hmmfrom  hmm to    alifrom  ali to    envfrom  env to     acc
 ---   ------ ----- --------- --------- -------- -------   -------- -------   -------- -------    ----
   1 !  150.7   0.0   2.8e-46   2.3e-41       14     204 ..       14     204 ..        4     212 .] 0.93

  Alignments for each domain:
  == domain 1  score: 150.7 bits;  conditional E-value: 2.8e-46
          AtPPa1_AF_a-0  14 PPPPtideikknnqflpvPphPwydfdrGsgapeitwvvilreeGarleyrldqqkGlvqlkrekqsptvdpfdeGfiPrtltelnkplltivvstlPvePGlwlkaeaiGllpvivlGlwnPiilavktedpnkrtiryanllkpqvliiiiehlrkrreqenkyvlvgpvlpaeeakeqilkaiimwei 204
                            P P   + i       +v  hPw+d++ G gap+i  vv+ + +G + +y+ld++ Gl++++r   s  v p++ Gf+Prtl e n p+  +v+ + Pv PG +l+a+aiGl+p+i +G ++  i+av ++dp+ ++ +  + l+p+ l  i++ ++  ++ enk v v ++lp+e+a e i+ ++  +
  sp|Q93V56|IPYR1_ARATH  14 PAPRLNERILSSLSRRSVAAHPWHDLEIGPGAPQIFNVVVEITKGSKVKYELDKKTGLIKVDRILYSSVVYPHNYGFVPRTLCEDNDPIDVLVIMQEPVLPGCFLRARAIGLMPMIDQGEKDDKIIAVCVDDPEYKHYTDIKELPPHRLSEIRRFFEDYKKNENKEVAVNDFLPSESAVEAIQYSMDLYAE 204
                            4455556677777777899*****************************************************************************************************************************************998887765 PP
```

```
Alignments for each domain:
== domain 1  score: 150.7 bits;  conditional E-value: 2.8e-46
        AtPPa1_AF_a-0  14 PPPPtideikknnqflpvPphPwydfdrGsgapeit
                          P P   + i       +v  hPw+d++ G gap+i
sp|Q93V56|IPYR1_ARATH  14 PAPRLNERILSSLSRRSVAAHPWHDLEIGPGAPQIF
                          4455556677777777899*****************
```

HMMer device

conicidences

DB sequence

Pay attention to:

- The sequence of interest should be in the top hits of HMMer
- The E-value should be small, REALLY SMALL
- The score should be at least 0.3 length of your sequence (sqL), *i.e.*for a 330 aa protein: Score > 99. PDB data give an average Score of 0.6 sqL. Rosetta GOOD predictions give a Score of 0.99 sqL. Be suspicious if value is too high.
- the alignment should be in frame start and end aminoacid numbers should match (correct the numbers if you truncated the sequence when modeling)
- There should be NO GAPS in your alignment!