PyMOL Command Reference

This is the list of all PyMOL commands which can be used in the PyMOL command line and in PML scripts. The command descriptions found in this file can also be printed to the PyMOL text buffer with the help command. Example:

PyMOL>help color ...

The list of arguments for a command (the "usage") can be queried on the command line with a questionmark. Example:

PyMOL>color \? Usage: color color [, selection [, quiet [, flags]]]

The square brackets ("[" and "]") indicate optional arguments and are not part of the syntax.

If the PyMOL command interpreter doesn't understand some input, it passes it to the Python interpreter. This means that single-line Python expressions can be put into PML scripts or typed into the command line. Prefixing a line with a slash (/) forces the interpreter to pass it to Python. See also the python command to input multi-line Python scripts.

This file can be generated on the PyMOL command line:

PyMOL>cmd.write_html_ref('pymol-command-ref.html')

- cd
- cealign
- center
- centerofmass
- check
- clean
- clip
- cls
- color
- color_deep
- conda
- config_mouse
- copy
- copy_to
- count_atoms
- count_discrete
- count_frames
- count_states
- create
- cycle_valence
- decline
- delete
- deprotect
- desaturate
- deselect
- diagnostics
- dihedral
- dir
- disable
- distance
- drag
- draw
- dss
- dump
- edit
- edit_mode
- embed
- enable
- ending

- extra_fit
- extract
- fab
- feedback
- fetch
- fit
- fix_chemistry
- flag
- fnab
- focal_blur
- fork
- forward
- fragment
- frame
- full_screen
- fuse
- get
- get_angle
- get_area
- get_bond
- get_chains
- get_dihedral
- get_distance
- get_extent
- get_position
- get_property
- get_property_list
- get_renderer
- get_sasa_relative
- get_symmetry
- get_title
- get_type
- get_version
- get_view
- get_viewport
- gradient
- group
- h_add
- h_fill

- h_fix
- help
- help_setting
- hide
- id_atom
- identify
- index
- indicate
- intra_fit
- intra_rms
- intra_rms_cur
- invert
- isodot
- isolevel
- isomesh
- isosurface
- iterate
- iterate_state
- join_states
- label
- load
- load_embedded
- load_mtz
- load_png
- load_traj
- loadall
- log
- log_close
- log_open
- ls
- madd
- map_double
- map_halve
- map_new
- map_set
- map_set_border
- map_trim
- mappend
- mask

- matrix_copy
- matrix_reset
- mclear
- mcopy
- mdelete
- mdo
- mdump
- mem
- meter_reset
- middle
- minsert
- mmatrix
- mmove
- morph
- mouse
- move
- movie.load
- movie.nutate
- movie.pause
- movie.produce
- movie.rock
- movie.roll
- movie.screw
- movie.sweep
- movie.tdroll
- movie.zoom
- mplay
- mpng
- mse2met
- mset
- mstop
- mtoggle
- multifilesave
- multisave
- mview
- order
- orient
- origin
- overlap

- pair_fit
- phi_psi
- pi_interactions
- pip
- png
- pop
- protect
- pseudoatom
- pwd
- python
- quit
- ramp_new
- ramp_update
- ray
- rebond
- rebuild
- recolor
- redo
- reference
- refresh
- refresh_wizard
- reinitialize
- remove
- remove_picked
- rename
- replace
- replace_wizard
- reset
- resume
- rewind
- rms
- rms_cur
- rock
- rotate
- run
- save
- scene
- scene_order
- sculpt_activate

- sculpt_deactivate
- sculpt_iterate
- sculpt_purge
- select
- set
- set_atom_property
- set_bond
- set_color
- set_dihedral
- set_geometry
- set_key
- set_name
- set_property
- set_symmetry
- set_title
- set_view
- show
- skip
- slice_new
- smooth
- sort
- space
- spawn
- spectrum
- spheroid
- splash
- split_chains
- split_states
- stereo
- super
- symexp
- symmetry_copy
- system
- toggle
- torsion
- translate
- turn
- unbond
- undo

- ungroup
- uniquify
- unmask
- unpick
- unset
- unset_bond
- unset_deep
- update
- util.cbab
- util.cbac
- util.cbag
- util.cbak
- util.cbam
- util.cbao
- util.cbap
- util.cbas
- util.cbaw
- util.cbay
- util.cbc
- util.chainbow
- util.cnc
- util.rainbow
- util.ss
- valence
- vdw_fit
- view
- viewport
- volume
- volume_color
- volume_panel
- volume_ramp_new
- window
- wizard
- zoom
- *

abort

DESCRIPTION

```
"abort" abruptly terminates execution of the PyMOL command script
without executing any additional commands.
```

SEE ALSO

```
[embed](https://pymol.org/pymol-command-ref.html#embed), [skip]
(https://pymol.org/pymol-command-ref.html#skip), [python]
(https://pymol.org/pymol-command-ref.html#python)
```

api: pymol.helping.abort

---

accept

DESCRIPTION

```
"accept" is an internal method for handling of session file
security.
```

api: pymol.moving.accept

---

alias

DESCRIPTION

```
"alias" binds routinely-used command inputs to a new command
keyword.
```

USAGE

```
alias name, command
```

ARGUMENTS

```
name = string: new keyword

command = string: literal input with commands separated by semicolons.
```

EXAMPLE

```
alias my\_scene, hide; show ribbon, polymer; show sticks, organic; show
nonbonded, solvent

my\_scene
```

NOTES

```
For security reasons, aliased commands are not saved or restored
in sessions.
```

SEE ALSO

```
cmd.extend, [api](https://pymol.org/pymol-command-ref.html#api)
```

api: pymol.commanding.alias

---

align

DESCRIPTION

```
"align" performs a sequence alignment followed by a structural
superposition, and then carries out zero or more cycles of
refinement in order to reject structural outliers found during
the fit. "align" does a good job on proteins with decent sequence
similarity \(identity >30\%\). For comparing proteins with lower
sequence identity, the "super" and "cealign" commands perform
better.
```

USAGE

```
align mobile, target \[, cutoff \[, cycles
    \[, gap \[, extend \[, max\_gap \[, object
    \[, matrix \[, mobile\_state \[, target\_state
    \[, quiet \[, max\_skip \[, transform \[, reset \]\]\]\]\]\]\]\]\]\]\]\]\]\]\]\]\]
```

ARGUMENTS

```
mobile = string: atom selection of mobile object

target = string: atom selection of target object

cutoff = float: outlier rejection cutoff in Angstrom \{default: 2.0\}

cycles = int: maximum number of outlier rejection cycles \{default: 5\}

gap, extend, max\_gap: sequence alignment parameters

object = string: name of alignment object to create \{default: \(no
alignment object\)\}

matrix = string: file name of substitution matrix for sequence
alignment \{default: BLOSUM62\}

mobile\_state = int: object state of mobile selection \{default: 0 = all
states\}

target\_state = int: object state of target selection \{default: 0 = all
states\}

transform = 0/1: do superposition \{default: 1\}
```

NOTES

```
If object is specified, then align will create an object which
indicates paired atoms and supports visualization of the alignment
in the sequence viewer.

The RMSD of the aligned atoms \(after outlier rejection\!\) is reported
in the text output. The all-atom RMSD can be obtained by setting
cycles=0 and thus not doing any outlier rejection.
```

EXAMPLE

```
align protA////CA, protB////CA, object=alnAB
```

SEE ALSO

```
[super](https://pymol.org/pymol-command-ref.html#super), [cealign]
(https://pymol.org/pymol-command-ref.html#cealign), [pair\_fit]
(https://pymol.org/pymol-command-ref.html#pair_fit), [fit]
(https://pymol.org/pymol-command-ref.html#fit), [rms](https://pymol.org/pymol-
command-ref.html#rms), [rms\_cur](https://pymol.org/pymol-command-
ref.html#rms_cur), [intra\_rms](https://pymol.org/pymol-command-
ref.html#intra_rms), [intra\_rms\_cur](https://pymol.org/pymol-command-
ref.html#intra_rms_cur)
```

api: pymol.fitting.align

---

alignto

DESCRIPTION

```
    "alignto" aligns all other loaded objects to the target
    using the specified alignment algorithm.
```

USAGE

```
    alignto target \[, method \[, quiet \]\]
```

NOTES

```
    Available alignment methods are "align", "super" and "cealign".
```

EXAMPLE

```
    # fetch some calmodulins
    fetch 1cll 1sra 1ggz 1k95, async=0

    # align them to 1cll using cealign
    alignto 1cll, method=cealign
    alignto 1cll, object=all\_to\_1cll
```

SEE ALSO

```
    [extra\_fit](https://pymol.org/pymol-command-ref.html#extra_fit), [align]
(https://pymol.org/pymol-command-ref.html#align), [super]
(https://pymol.org/pymol-command-ref.html#super), [cealign]
(https://pymol.org/pymol-command-ref.html#cealign), [fit]
(https://pymol.org/pymol-command-ref.html#fit), [rms](https://pymol.org/pymol-
command-ref.html#rms), [rms\_cur](https://pymol.org/pymol-command-
ref.html#rms_cur), [intra\_fit](https://pymol.org/pymol-command-
ref.html#intra_fit)
```

api: pymol.fitting.alignto

---

alphatoall

DESCRIPTION

```
Expand any given property of the CA atoms to all atoms in the residue
```

ARGUMENTS

```
selection = string: atom selection \{default: polymer\}

properties = string: space separated list of atom properties \{default: b\}
```

api: pymol.editing.alphatoall

---

alter

DESCRIPTION

```
"alter" changes atomic properties using an expression evaluated
within a temporary namespace for each atom.
```

USAGE

```
alter selection, expression
```

EXAMPLES

```
alter chain A, chain='B'
alter all, resi=str\(int\(resi\)+100\)
sort
```

NOTES

```
Symbols defined \(\* = read only\):

name, resn, resi, resv, chain, segi, elem, alt, q, b, vdw, type,
partial\_charge, formal\_charge, elec\_radius, text\_type, label,
numeric\_type, model\*, state\*, index\*, ID, rank, color, ss,
cartoon, flags
```

```
All strings must be explicitly quoted.  This operation typically
takes several seconds per thousand atoms altered.

You may need to issue a "rebuild" in order to update associated
representations.

WARNING: You should always issue a "sort" command on an object
after modifying any property which might affect canonical atom
ordering \(names, chains, etc.\).  Failure to do so will confound
subsequent "create" and "byres" operations.
```

SEE ALSO

```
[alter\_state](https://pymol.org/pymol-command-ref.html#alter_state), [iterate]
(https://pymol.org/pymol-command-ref.html#iterate), [iterate\_state]
(https://pymol.org/pymol-command-ref.html#iterate_state), [sort]
(https://pymol.org/pymol-command-ref.html#sort)
```

api: pymol.editing.alter

---

alter_state

DESCRIPTION

```
"alter\_state" changes atom coordinates and flags over a particular
state and selection using the Python evaluator with a temporary
namespace for each atomic coordinate.
```

USAGE

```
alter\_state state, selection, expression
```

EXAMPLES

```
alter\_state 1, all, x=x+5
rebuild
```

NOTES

```
By default, most of the symbols from "alter" are available for use
on a read-only basis.

It is usually necessary to "rebuild" representations once your
alterations are complete.
```

SEE ALSO

```
[iterate\_state](https://pymol.org/pymol-command-ref.html#iterate_state), [alter]
(https://pymol.org/pymol-command-ref.html#alter), [iterate]
(https://pymol.org/pymol-command-ref.html#iterate)
```

api: pymol.editing.alter_state

angle

DESCRIPTION

```
"angle" shows the angle formed between any three atoms.
```

USAGE

```
angle \[ name \[, selection1 \[, selection2 \[, selection3 \]\]\]\]
```

NOTES

```
"angle" alone will show the angle angle formed by selections \(pk1\),
\(pk2\), \(pk3\) which can be set using the "PkAt" mouse action
\(typically, Ctrl-middle-click\)
```

PYMOL API

```
cmd.angle\(string name, string selection1, string selection2,
          string selection3\)
```

SEE ALSO

```
[distance](https://pymol.org/pymol-command-ref.html#distance), [dihedral]
(https://pymol.org/pymol-command-ref.html#dihedral)
```

api: pymol.querying.angle

api

DESCRIPTION

```
API helper function. Get the full function name \(incl. module\) of
given command.
```

ARGUMENTS

```
name = string: name of a PyMOL command
```

NOTES

```
The PyMOL Python Application Programming Interface \(API\) should be
accessed exclusively through the "cmd" module \(never "\_cmd"\!\).  Nearly
all command-line functions have a corresponding API method.

from pymol import cmd
result = cmd.\<command-name>\( argument , ... \)

Although the PyMOL core is not multi-threaded, the API is
thread-safe and can be called asynchronously by external python
programs.  PyMOL handles the necessary locking to insure that
internal states do not get corrupted.  This makes it very easy to
build complicated systems which involve direct realtime visualization.
```

api: pymol.helping.api

---

as

DESCRIPTION

```
"as" turns on and off atom and bond representations.
```

USAGE

```
as representation \[, selection \]
```

ARGUMENTS

```
representation = lines, spheres, mesh, ribbon, cartoon, sticks,
    dots, surface, labels, extent, nonbonded, nb\_spheres, slice,
    extent, slice, dashes, angles, dihedrals, cgo, cell, callback,
    volume or everything

selection = string \{default: all\}
```

EXAMPLES

```
as lines, name CA+C+N

as ribbon
```

PYMOL API

```
cmd.show\_as\(string representation, string selection\)
```

NOTES

```
"selection" can be an object name
"as" alone will turn on lines and nonbonded and hide everything else.
```

SEE ALSO

```
[show](https://pymol.org/pymol-command-ref.html#show), [hide]
(https://pymol.org/pymol-command-ref.html#hide), [enable]
(https://pymol.org/pymol-command-ref.html#enable), [disable]
(https://pymol.org/pymol-command-ref.html#disable)
```

api: pymol.viewing.show_as

---

assign_stereo

DESCRIPTION

```
Assign "stereo" atom property \(R/S stereochemistry\).

Requires either a Schrodinger Suite installation \(SCHRODINGER
environment variable set\) or RDKit \(rdkit Python module\).
```

USAGE

```
assign\_stereo \[selection \[, state \[, method \]\]\]
```

ARGUMENTS

```
selection = str: atom selection \{default: all\}

state = int: object state \{default: -1 \(current\)\}

method = schrodinger or rdkit: \{default: try both\}
```

api: pymol.stereochemistry.assign_stereo

---

attach

DESCRIPTION

```
"attach" adds a single atom on to the picked atom.
```

USAGE

```
attach element, geometry, valence
```

PYMOL API

```
cmd.attach\( element, geometry, valence \)
```

api: pymol.editing.attach

---

backward

DESCRIPTION

```
"backward" moves the movie back one frame.
```

USAGE

```
backward
```

PYMOL API

```
cmd.backward\(\)
```

SEE ALSO

```
[mset](https://pymol.org/pymol-command-ref.html#mset), [forward]
(https://pymol.org/pymol-command-ref.html#forward), [rewind]
(https://pymol.org/pymol-command-ref.html#rewind)
```

api: pymol.moving.backward

---

bg_color

DESCRIPTION

```
"bg\_color" sets the background color.
```

USAGE

```
bg\_color \[ color \]
```

ARGUMENTS

```
color = string: color name or number \{default: black\}
```

EXAMPLES

```
bg\_color grey30

bg\_color
```

NOTES

```
To obtain a transparent background, "unset opaque\_background", and
then use "ray".
```

SEE ALSO

```
[set\_color](https://pymol.org/pymol-command-ref.html#set_color), [ray]
(https://pymol.org/pymol-command-ref.html#ray)
```

PYMOL API

```
cmd.bg\_color\(string color\)
```

api: pymol.viewing.bg_color

---

bond

DESCRIPTION

```
"bond" creates a new bond between two selections, each of which
should contain one atom.
```

USAGE

```
bond \[atom1, atom2 \[,order\]\]
```

NOTES

```
The atoms must both be within the same object.

The default behavior is to create a bond between the \(lb\) and \(rb\)
selections.
```

PYMOL API

```
cmd.bond\(string atom1, string atom2\)
```

SEE ALSO

```
[unbond](https://pymol.org/pymol-command-ref.html#unbond), [fuse]
(https://pymol.org/pymol-command-ref.html#fuse), [attach]
(https://pymol.org/pymol-command-ref.html#attach), [replace]
(https://pymol.org/pymol-command-ref.html#replace), [remove\_picked]
(https://pymol.org/pymol-command-ref.html#remove_picked)
```

api: pymol.editing.bond

---

button

DESCRIPTION

```
"button" can be used to redefine what the mouse buttons do.
```

USAGE

```
button button, modifier, action
```

ARGUMENTS

```
button = left, middle, right, wheel, double\_left, double\_middle,
    double\_right, single\_left, single\_middle, or single\_right

modifiers = None, Shft, Ctrl, CtSh, CtAl, CtAl, CtAS,

actions = None, Rota, Move, MovZ, Slab, +Box, -Box, Clip, MovS,
    +/-, PkAt, Pk1, MvSZ, Sele, Orig, Menu, PkAt, Pk1 RotO, MovO,
    MvOZ, MovA, PkAt, PkTB, MvSZ MvAZ, DrgM, RotZ, PkBd, ClpN,
    ClpF
```

NOTES

Changes made using the button command are easily overridden when the user iterates through the mouse modes. This behavior needs to be changed.

Obsolete actions: lb, mb, rb, +lb, +mb, +rb, +lbX, -lbX,
Unsupported, Internal, or Future Actions: RotD, MovD, MvDZ, RotF, MovF, MvFZ, TorF, RotV, MovV, MvVZ, DgMZ, DgRT

PYMOL API

```
cmd.button\(string button, string modifier, string action\)
```

SEE ALSO

```
[config\_mouse](https://pymol.org/pymol-command-ref.html#config_mouse)
```

api: pymol.controlling.button

---

cache

DESCRIPTION

```
"cache" manages storage of precomputed results, such as
molecular surfaces.
```

USAGE

```
cache action \[, scenes \[, state \]\]
```

ARGUMENTS

```
action = string: enable, disable, read\_only, clear, or optimize

scenes = string: a space-separated list of scene names \(default: ''\)

state = integer: state index \(default: -1\)
```

EXAMPLES

```
cache enable
cache optimize
cache optimize, F1 F2 F5
```

NOTES

```
"cache optimize" will iterate through the list of scenes provided
\(or all defined scenes\), compute any missing surfaces, and store
them in the cache for later reuse.
```

PYMOL API

```
cmd.cache\(string action, string scenes, int state, int quiet\)
```

api: pymol.exporting.cache

---

callout

DESCRIPTION

```
Create a new screen-stabilized callout object.
```

ARGUMENTS

```
name = str: object name

label = str: label text

pos = str or list: anchor in model space as 3-float coord list or atom
selection. If empty, don't draw an arrow. \{default: \}

screen = str or list: position on screen as 2-float list between \[-1,-1\]
\(lower left\) and \[1,1\] \(upper right\) or "auto" for smart placement.
\{default: auto\}
```

api: pymol.experimenting.callout

---

capture

UNDOCUMENTED

api: pymol.viewing.capture

---

cartoon

DESCRIPTION

```
"cartoon" changes the default cartoon representation for a set of atoms.
```

USAGE

```
cartoon type, selection
```

ARGUMENTS

```
type = automatic, skip, loop, rectangle, oval, tube, arrow, dumbbell
```

PYMOL API

```
cmd.cartoon\(string type, string selection\)
```

EXAMPLES

```
cartoon rectangle, chain A

cartoon skip, resi 145-156
```

NOTES

```
This command is rarely required since the default "automatic" mode
chooses cartoons according to the information in the PDB HELIX and
SHEET records.
```

api: pymol.viewing.cartoon

---

cd

DESCRIPTION

```
"cd" changes the current working directory.
```

USAGE

```
cd \<path>
```

SEE ALSO

```
[pwd](https://pymol.org/pymol-command-ref.html#pwd), [ls]
(https://pymol.org/pymol-command-ref.html#ls), [system](https://pymol.org/pymol-
command-ref.html#system)
```

api: pymol.externing.cd

---

cealign

DESCRIPTION

```
"cealign" aligns two proteins using the CE algorithm.
```

USAGE

```
cealign target, mobile \[, target\_state \[, mobile\_state \[, quiet \[,
    guide \[, d0 \[, d1 \[, window \[, gap\_max, \[, transform
\]\]\]\]\]\]\]\]\]
```

NOTES

```
If "guide" is set PyMOL will align using only alpha carbons, which is the
default behavior. Otherwise, PyMOL will use all atoms. If "quiet" is set
to -1, PyMOL will print the rotation matrix as well.

Reference: Shindyalov IN, Bourne PE \(1998\) Protein structure alignment by
incremental combinatorial extension \(CE\) of the optimal path.  Protein
Engineering 11\(9\) 739-747.
```

EXAMPLES

```
cealign protA////CA, protB////CA

# fetch two proteins and align them
fetch 1rlw 1rsy, async=0
cealign 1rlw, 1rsy
```

SEE ALSO

```
[align](https://pymol.org/pymol-command-ref.html#align), [pair\_fit]
(https://pymol.org/pymol-command-ref.html#pair_fit), [fit]
(https://pymol.org/pymol-command-ref.html#fit), [rms](https://pymol.org/pymol-
command-ref.html#rms), [rms\_cur](https://pymol.org/pymol-command-
ref.html#rms_cur), [intra\_rms](https://pymol.org/pymol-command-
ref.html#intra_rms), [intra\_rms\_cur](https://pymol.org/pymol-command-
ref.html#intra_rms_cur), [super](https://pymol.org/pymol-command-ref.html#super)
```

api: pymol.fitting.cealign

---

center

DESCRIPTION

```
"center" translates the window, the clipping slab, and the
origin to a point centered within the atom selection.
```

USAGE

```
center \[ selection \[, state \[, origin \[, animate \]\]\]\]
```

EXAMPLES

```
center chain B
center 145/
```

ARGUMENTS

```
selection = string: selection-expression or name pattern \(default: "all"\).

state = 0 \(default\) use all coordinate states

state = -1 use only coordinates for the current state

state > 0  use coordinates for a specific state

origin = 1 \(default\) move the origin

origin = 0 leave the origin unchanged
```

PYMOL API

```
cmd.center\(string selection, int state, int origin\)
```

SEE ALSO

```
[origin](https://pymol.org/pymol-command-ref.html#origin), [orient]
(https://pymol.org/pymol-command-ref.html#orient), [zoom]
(https://pymol.org/pymol-command-ref.html#zoom)
```

api: pymol.viewing.center

---

centerofmass

DESCRIPTION

```
Calculates the center of mass. Considers atom mass and occupancy.
```

ARGUMENTS

```
selection = string: atom selection \{default: all\}

state = integer: object state, -1 for current state, 0 for all states
\{default: -1\}
```

NOTES

```
If occupancy is 0.0 for an atom, set it to 1.0 for the calculation
\(assume it was loaded from a file without occupancy information\).
```

SEE ALSO

```
[get\_extent](https://pymol.org/pymol-command-ref.html#get_extent)
```

api: pymol.querying.centerofmass

---

check

DESCRIPTION

```
"check" is unsupported command that may eventually have something
to do with assigning forcefield parameters to a selection of
atoms.
```

api: pymol.experimenting.check

---

clean

DESCRIPTION

```
Note: This operation is limited to 999 atoms.

Run energy minimization on the given selection, using an MMFF94
force field.
```

ARGUMENTS

```
selection = str: atom selection to minimize

present = str: selection of fixed atoms to restrain the minimization

state = int: object state \{default: -1 \(current\)\}

fix = UNUSED

restraing = UNUSED

method = UNUSED

async = 0/1: run in separate thread \{default: 0\}

save\_undo = UNUSED

message = Message to display during async minimization
```

EXAMPLE

```
# minimize ligand in binding pocket
clean organic, all within 8 of organic
```

api: pymol.computing.clean

---

clip

DESCRIPTION

```
"clip" alters the positions of the clipping planes.
```

USAGE

```
clip mode, distance \[, selection \[, state \]\]
```

ARGUMENTS

```
mode = near, far, move, slab, or atoms

distance is a floating point value

selection = atom selection \(for mode=atoms only\)
```

EXAMPLES

```
clip near, -5           # moves near plane away from you by 5 A
clip far, 10            # moves far plane towards you by 10 A
clip move, -5           # moves the slab away from you by 5 A
clip slab, 20           # sets slab thickness to 20 A
clip slab, 10, resi 11  # clip 10 A slab about residue 11

clip atoms, 5, pept     # clip atoms in "pept" with a 5 A buffer
                        # about their current camera positions
```

PYMOL API

```
cmd.clip\(string mode, float distance, string selection, int state\)
```

SEE ALSO

```
[zoom](https://pymol.org/pymol-command-ref.html#zoom), [orient]
(https://pymol.org/pymol-command-ref.html#orient), [reset]
(https://pymol.org/pymol-command-ref.html#reset)
```

api: pymol.viewing.clip

---

cls

DESCRIPTION

```
"cls" clears the output buffer.
```

USAGE

```
cls
```

api: pymol.commanding.cls

---

color

DESCRIPTION

```
"color" changes the color of objects or atoms.
```

USAGE

```
color color \[, selection \]
```

ARGUMENTS

```
color = string: color name or number

selection = string: selection-expression or name-pattern
corresponding to the atoms or objects to be colored
\{default: \(all\)\}.
```

NOTES

```
When using color ramps, the ramp can be used as a color.
```

PYMOL API

```
cmd.color\(string color, string selection, int quiet\)
```

SEE ALSO

```
[color\_deep](https://pymol.org/pymol-command-ref.html#color_deep), [set\_color]
(https://pymol.org/pymol-command-ref.html#set_color), [recolor]
(https://pymol.org/pymol-command-ref.html#recolor)
```

EXAMPLE

```
color cyan
color yellow, chain A
```

api: pymol.viewing.color

---

color_deep

DESCRIPTION

```
Unset all object and atom level \(not global\) color settings and
apply given color.
```

ARGUMENTS

```
color = str: color name or number

name = str: object name or pattern \{default: all\}
```

SEE ALSO

```
[color](https://pymol.org/pymol-command-ref.html#color), [unset\_deep]
(https://pymol.org/pymol-command-ref.html#unset_deep)
```

api: pymol.viewing.color_deep

---

conda

DESCRIPTION

```
Experimental and limited conda wrapper for PyMOL bundles. Automatically
passes "--yes" to "conda install".
```

EXAMPLE

```
conda install biopython
```

api: pymol.externing.conda

---

config_mouse

DESCRIPTION

```
"config\_mouse" sets the current mouse configuration ring.
```

USAGE

```
config\_mouse ring
```

EXAMPLES

```
config\_mouse three\_button
config\_mouse two\_button
config\_mouse one\_button
```

PYMOL API

```
cmd.config\_mouse\(string ring, int quiet\)
```

SEE ALSO

```
[mouse](https://pymol.org/pymol-command-ref.html#mouse), [button]
(https://pymol.org/pymol-command-ref.html#button)
```

api: pymol.controlling.config_mouse

---

copy

DESCRIPTION

```
"copy" creates a new object that is an identical copy of an
existing object.
```

USAGE

```
copy target, source \[, zoom \]
```

NOTES

```
Currently, this command only works for molecular objects.
```

SEE ALSO

```
[create](https://pymol.org/pymol-command-ref.html#create)
```

api: pymol.creating.copy

---

copy_to

DESCRIPTION

```
Copies selection to object \`name\` \(all states\) and by default
renames chain, segi and ID identifiers to avoid naming conflicts.
```

ARGUMENTS

```
name = str: object name to modify

selection = str: atom selection \(will be copied to \`name\`\)

rename = str: space separated list of identifiers to rename
\{default: chain segi ID\}
```

SEE ALSO

```
[create](https://pymol.org/pymol-command-ref.html#create), [fuse]
(https://pymol.org/pymol-command-ref.html#fuse)
```

api: pymol.editing.copy_to

---

count_atoms

DESCRIPTION

```
"count\_atoms" returns a count of atoms in a selection.
```

USAGE

```
count\_atoms \[ selection \[, quiet \[, state \]\]\]
```

api: pymol.querying.count_atoms

---

count_discrete

DESCRIPTION

```
Count the number of discrete objects in selection.
```

USAGE

```
count\_discrete selection
```

api: pymol.querying.count_discrete

---

count_frames

DESCRIPTION

```
"count\_frames" returns the number of frames defined for the PyMOL
movie.
```

USAGE

```
count\_frames
```

PYMOL API

```
cmd.count\_frames\(\)
```

SEE ALSO

```
[frame](https://pymol.org/pymol-command-ref.html#frame), [count\_states]
(https://pymol.org/pymol-command-ref.html#count_states)
```

api: pymol.querying.count_frames

---

count_states

DESCRIPTION

```
"count\_states" returns the number of states in the selection.
```

USAGE

```
count\_states
```

PYMOL API

```
cmd.count\_states\(string selection\)
```

SEE ALSO

```
[frame](https://pymol.org/pymol-command-ref.html#frame)
```

api: pymol.querying.count_states

---

create

DESCRIPTION

```
"create" creates a new molecule object from a selection.  It can
also be used to create states in an existing object.
```

USAGE

```
create name, selection \[,source\_state \[,target\_state \] \]
```

ARGUMENTS

```
name = string: name of object to create or modify

selection = string: atoms to include in the new object

source\_state = integer: \{default: 0 -- copy all states\}

target\_state = integer: -1 appends after last state \{default: 0\}

copy\_properties = boolean: \{default: False\} copies properties from source
                           \(last of the same properties survives with multiple
objects\)
```

PYMOL API

```
cmd.create\(string name, string selection, int state,
           int target\_state, int discrete\)
```

NOTES

```
If the source and target states are zero \(default\), then all
states will be copied.  Otherwise, only the indicated states will
be copied.
```

SEE ALSO

```
[load](https://pymol.org/pymol-command-ref.html#load), [copy]
(https://pymol.org/pymol-command-ref.html#copy), [extract]
(https://pymol.org/pymol-command-ref.html#extract)
```

api: pymol.creating.create

---

cycle_valence

DESCRIPTION

```
"cycle\_valence" cycles the valence on the currently selected bond.
```

USAGE

```
cycle\_valence \[ h\_fill \]
```

ARGUMENTS

```
h\_fill = 0 or 1: updated hydrogens too\? \{default: 1 \(yes\)\}
```

EXAMPLE

```
cycle\_valence
```

NOTES

```
If the h\_fill flag is true, hydrogens will be added or removed to
satisfy valence requirements.

This function is usually connected to the DELETE key and "CTRL-W".
```

PYMOL API

```
cmd.cycle\_valence\(int h\_fill\)
```

SEE ALSO

```
[remove\_picked](https://pymol.org/pymol-command-ref.html#remove_picked),
[attach](https://pymol.org/pymol-command-ref.html#attach), [replace]
(https://pymol.org/pymol-command-ref.html#replace), [fuse]
(https://pymol.org/pymol-command-ref.html#fuse), [h\_fill]
(https://pymol.org/pymol-command-ref.html#h_fill)
```

api: pymol.editing.cycle_valence

---

decline

DESCRIPTION

```
"decline" is an internal method for handling of session file
security.
```

api: pymol.moving.decline

---

delete

DESCRIPTION

```
"delete" removes objects and named selections
```

USAGE

```
delete name
```

ARGUMENTS

```
name = name\(s\) of object\(s\) or selection\(s\), supports wildcards \(\*\)
```

EXAMPLES

```
delete measure\*     # delete all objects which names start with "measure"
delete all          # delete all objects and selections
```

PYMOL API

```
cmd.delete \(string name = object-or-selection-name \)
```

SEE ALSO

```
[remove](https://pymol.org/pymol-command-ref.html#remove)
```

api: pymol.commanding.delete

---

deprotect

DESCRIPTION

```
"deprotect" reverses the effect of the "protect" command.
```

USAGE

```
deprotect \(selection\)
```

PYMOL API

```
cmd.deprotect\(string selection\)
```

SEE ALSO

```
[protect](https://pymol.org/pymol-command-ref.html#protect), [mask]
(https://pymol.org/pymol-command-ref.html#mask), [unmask]
(https://pymol.org/pymol-command-ref.html#unmask), [mouse]
(https://pymol.org/pymol-command-ref.html#mouse), editing
```

api: pymol.editing.deprotect

---

desaturate

DESCRIPTION

```
Desaturate the colors in the given selection.
```

ARGUMENTS

```
selection = str: atom selection \{default: all\}

a = float \[0..1\]: desaturation factor \{default: 0.5\}
```

api: pymol.experimenting.desaturate

---

deselect

DESCRIPTION

```
"deselect" disables any and all visible selections
```

USAGE

```
deselect
```

PYMOL API

```
cmd.deselect\(\)
```

api: pymol.selecting.deselect

---

diagnostics

DESCRIPTION

```
Get system level diagnostics data
```

USAGE

```
diagnostics \[ filename \]
```

ARGUMENTS

```
filename = str: If given, write output to text file
```

api: pymol.diagnosing.diagnostics

---

dihedral

DESCRIPTION

```
"dihedral" shows dihedral angles formed between any four atoms.
```

USAGE

```
dihedral \[ name \[, selection1 \[, selection2 \[, selection3 \[, selection4
\]\]\]\]\]
```

NOTES

```
"dihedral" alone will show the dihedral angle formed by selections
\(pk1\), \(pk2\), \(pk3\), and \(pk4\), which can be set using the "PkAt"
mouse action \(typically, Ctrl-middle-click\)
```

PYMOL API

```
cmd.dihedral\(string name, string selection1, string selection2,
             string selection3, string selection4\)
```

SEE ALSO

```
[distance](https://pymol.org/pymol-command-ref.html#distance), [angle]
(https://pymol.org/pymol-command-ref.html#angle)
```

api: pymol.querying.dihedral

---

dir

DESCRIPTION

```
List contents of the current working directory.
```

USAGE

```
ls \[pattern\]
dir \[pattern\]
```

EXAMPLES

```
ls
ls \*.pml
```

SEE ALSO

```
[cd](https://pymol.org/pymol-command-ref.html#cd), [pwd](https://pymol.org/pymol-
command-ref.html#pwd), [system](https://pymol.org/pymol-command-ref.html#system)
```

api: pymol.externing.ls

---

disable

DESCRIPTION

```
"disable" turns off display of one or more objects and/or selections.
```

USAGE

```
disable name
```

ARGUMENTS

```
name = name-pattern or selection.
```

PYMOL API

```
cmd.disable\(string name\)
```

SEE ALSO

```
[show](https://pymol.org/pymol-command-ref.html#show), [hide]
(https://pymol.org/pymol-command-ref.html#hide), [enable]
(https://pymol.org/pymol-command-ref.html#enable)
```

api: pymol.viewing.disable

---

distance

DESCRIPTION

```
"distance" creates a new distance object between two selections.
```

USAGE

```
distance \[name \[, selection1 \[, selection2 \[, cutoff \[, mode \]\]\]\]\]
```

ARGUMENTS

```
name = string: name of the distance object to create

selection1 = string: first atom selection

selection2 = string: second atom selection

cutoff = float: longest distance to show

mode = 0: all interatomic distances

mode = 1: only bond distances

mode = 2: only show polar contact distances
```

```
mode = 3: like mode=0, but use distance\_exclusion setting

mode = 4: distance between centroids \(does not support
          dynamic\_measures; new in PyMOL 1.8.2\)

mode = 5: pi-pi and pi-cation interactions

mode = 6: pi-pi interactions

mode = 7: pi-cation interactions

mode = 8: like mode=3, but cutoff is the ratio between
          distance and sum of VDW radii

state = int: object state to create the measurement object in
and to get coordinates from \{default: 0 \(all states\)\}

state1, state2 = int: overrule 'state' argument to measure distances
between different states \{default: use state\}
```

EXAMPLES

```
distance mydist, 14/CA, 29/CA

distance hbonds, all, all, 3.2, mode=2
```

NOTES

```
The distance wizard makes measuring distances easier than using
the "dist" command for real-time operations.

"dist" alone will show distances between selections \(pk1\) and \(pk1\),
which can be set using the PkAt mouse action \(usually CTRL-middle-click\).
```

PYMOL API

```
cmd.distance\(string name, string selection1, string selection2,
             string cutoff, string mode \)
```

api: pymol.querying.distance

---

drag

DESCRIPTION

```
"drag" activates dragging for a selection, enabling the user to
manipulate the atom coordinates of the atoms using mouse controls
similar to those for controlling the camera.
```

USAGE

```
drag \[ selection \]
```

ARGUMENTS

```
selection = string: atoms to drag.  If not provided, and dragging
is active, then dragging is instead deactivated.
```

NOTES

```
Currently, the selection of atom to drag must all reside in a
single molecular object.
```

api: pymol.editing.drag

---

draw

DESCRIPTION

```
"draw" creates an OpenGL-based image of the current frame.
```

USAGE

```
draw \[width \[,height \[,antialias \]\]\]
```

ARGUMENTS

```
width = integer \{default: 0 \(current\)\}

height = integer \{default: 0 \(current\)\}

antialias = integer \{default: -1 \(use antialias setting\)\}
```

EXAMPLES

```
draw
draw 1600
```

NOTES

```
Default width and height are taken from the current viewpoint. If
one is specified but not the other, then the missing value is
scaled so as to preserve the current aspect ratio.

Because this feature uses the OpenGL rendering context to piece
together the image, it does not work when running in the
command-line only mode.

On certain graphics hardware, "unset opaque\_background" followed
by "draw" will produce an image with a transparent background.
However, better results can usually be obtained using "ray".
```

PYMOL API

```
cmd.draw\(int width, int height, int antialias, int quiet\)
```

SEE ALSO

```
[ray](https://pymol.org/pymol-command-ref.html#ray), [png]
(https://pymol.org/pymol-command-ref.html#png), [save](https://pymol.org/pymol-
command-ref.html#save)
```

api: pymol.viewing.draw

---

dss

DESCRIPTION

```
"dss" defines secondary structure based on backbone geometry
and hydrogen bonding patterns.
```

USAGE

```
dss selection, state
```

ARGUMENT

```
selection = string: \{default: \(all\)\}

state = integer: \{default: 0 -- all states\}
```

EXAMPLE

```
dss
```

NOTES

```
With PyMOL, heavy emphasis is placed on cartoon aesthetics, and so
both hydrogen bonding patterns and backbone geometry are used in
the assignment process.  Depending upon the local context, helix
and strand assignments are made based on geometry, hydrogen
bonding, or both.

This command will generate results which differ slightly from DSSP
and other programs.  Most deviations occur in borderline or
transition regions.  Generally speaking, PyMOL is more strict, thus
assigning fewer helix/sheet residues, except for partially
distorted helices, which PyMOL tends to tolerate.

WARNING: This algorithm has not yet been rigorously validated.

If you dislike one or more of the assignments made by dss, you can
use the alter command to make changes \(followed by "rebuild"\).
For example:

    alter 123-125/, ss='L'
```

```
    alter pk1, ss='S'
    alter 90/, ss='H'
    rebuild
```

PYMOL API

```
cmd.dss\(string selection, int state\)
```

api: pymol.editing.dss

---

dump

DESCRIPTION

```
The dump command writes the geometry of an isosurface, isomesh,
isodot, or map object to a simple text file. Each line contains one
vertex in case of representations, or one grid point in case of a map.

For surface objects, XYZ coordinates and the normal are exported.
Three lines make one triangle \(like GL\_TRIANGLES\).

For mesh objects, XYZ coordinates are exported \(no normals\).
The vertices form line strips \(like GL\_LINE\_STRIP\), a blank
line starts a new strip.

For dot objects, XYZ coordinates are exported.

For map objects, XYZ coordinates and the value at the point are
exported. This forms a grid map.
```

USAGE

```
dump filename, object, state=1, quiet=1
```

ARGUMENTS

```
filename = str: file that will be written
object = str: object name
```

EXAMPLE

```
fetch 1ubq, mymap, type=2fofc, async=0

dump gridmap.txt, mymap

isosurface mysurface, mymap
dump surfacegeometry.txt, mysurface

isomesh mymesh, mymap
dump meshgeometry.txt, mymesh

isodot mydot, mymap, quiet=1
dump dotgeometry.txt, mydot
```

SEE ALSO

```
COLLADA export
```

api: pymol.experimenting.dump

---

edit

DESCRIPTION

```
"edit" picks atoms or a bond for editing.
```

USAGE

```
edit selection1 \[, selection2 \[, selection3 \[, selection4 \[, pkresi \[,
pkbond \]\]\]\]\]
```

NOTES

```
If only one selection is provided, an atom is picked.

If two selections are provided, the bond between them
is picked \(by default, if one exists\).
```

PYMOL API

```
cmd.edit\(string selection1, string selection2,
         string selection3, string selection4,
         int pkresi, int pkbond, int quiet\)
```

SEE ALSO

```
[unpick](https://pymol.org/pymol-command-ref.html#unpick), [remove\_picked]
(https://pymol.org/pymol-command-ref.html#remove_picked), [cycle\_valence]
(https://pymol.org/pymol-command-ref.html#cycle_valence), [torsion]
(https://pymol.org/pymol-command-ref.html#torsion)
```

api: pymol.editing.edit

---

edit_mode

DESCRIPTION

```
"edit\_mode" switches the mouse into editing mode, if such a mode
is available in the current mouse ring.
```

api: pymol.controlling.edit_mode

---

embed

DESCRIPTION

```
"embed" delimits a block of data embedded in a PyMOL command
script.
```

USAGE

```
embed key \[, type \[, sentinel \]\]
```

ARGUMENTS

```
key = string: unique indentifier for the data

type = pdb, mol, mol2, sdf, xplor

sentinel = string: a unique string signalling the end of the data \{default:
embed end\}
```

EXAMPLE

```
embed wats, pdb
HETATM    1  O   WAT     1       2.573  -1.034  -1.721
HETATM    2  H1  WAT     1       2.493  -1.949  -1.992
HETATM    3  H2  WAT     1       2.160  -0.537  -2.427
HETATM    4  O   WAT     2       0.705   0.744   0.160
HETATM    5  H1  WAT     2      -0.071   0.264   0.450
HETATM    6  H2  WAT     2       1.356   0.064  -0.014
embed end
```

NOTES

```
Only text data formats can be used with embed
```

SEE ALSO

```
[abort](https://pymol.org/pymol-command-ref.html#abort), [skip]
(https://pymol.org/pymol-command-ref.html#skip), [python]
(https://pymol.org/pymol-command-ref.html#python)
```

api: pymol.helping.embed

---

enable

DESCRIPTION

```
"enable" turns on display of one or more objects and/or selections.
```

USAGE

```
enable name
```

ARGUMENTS

```
name = name-pattern or selection.
```

NOTES

```
If name matches a selection name, then selection indicator dots
are shown for atoms in that selection.  If name is a
selection-expression, then all objects with atoms in that
selection are enabled.

For an object's content to be displayed in the 3D viewer, the
object must be enabled AND at least one of the available
representations must be shown.
```

PYMOL API

```
cmd.enable\(string object-name\)
```

EXAMPLES

```
enable target\_protein  # enables the target\_protein object

enable 1dn2.\*   # enables all entities starting with 1dn2.

enable \*lig     # enables all entities ending with lig
```

SEE ALSO

```
[show](https://pymol.org/pymol-command-ref.html#show), [hide]
(https://pymol.org/pymol-command-ref.html#hide), [disable]
(https://pymol.org/pymol-command-ref.html#disable)
```

api: pymol.viewing.enable

---

ending

DESCRIPTION

```
"ending" goes to the end of the movie.
```

USAGE

```
ending
```

PYMOL API

```
cmd.ending\(\)
```

api: pymol.moving.ending

---

extra_fit

DESCRIPTION

```
Like "intra\_fit", but for multiple objects instead of
multiple states.
```

ARGUMENTS

```
selection = string: atom selection of multiple objects \{default: all\}

reference = string: reference object name \{default: first object in selection\}

method = string: alignment method \(command that takes "mobile" and "target"
arguments, like "align", "super", "cealign" \{default: align\}

... extra arguments are passed to "method"
```

SEE ALSO

```
[align](https://pymol.org/pymol-command-ref.html#align), [super]
(https://pymol.org/pymol-command-ref.html#super), [cealign]
(https://pymol.org/pymol-command-ref.html#cealign), [intra\_fit]
(https://pymol.org/pymol-command-ref.html#intra_fit), util.mass\_align
```

api: pymol.fitting.extra_fit

---

extract

DESCRIPTION

```
"extract" is simply a shorthand way calling the "create" command
with the extract argument activated, so that atoms in the new
object are removed from the source object.
```

USAGE

```
extract name, selection \[, source\_state \[, target\_state \]\]
```

SEE ALSO

```
[create](https://pymol.org/pymol-command-ref.html#create)
```

api: pymol.creating.extract

---

fab

DESCRIPTION

```
Build a peptide
```

ARGUMENTS

```
input = str: sequence in one-letter code

name = str: name of object to create \{default: \}

ss = int: Secondary structure 1=alpha helix, 2=antiparallel beta, 3=parallel
beta, 4=flat
```

EXAMPLE

```
fab ACDEFGH
fab ACDEFGH, helix, ss=1
```

api: pymol.editor.fab

---

feedback

DESCRIPTION

```
"feedback" changes the amount of information output by pymol.
```

USAGE

```
feedback action, module, mask
```

ARGUMENTS

```
action = set, enable, or disable

module = string: a space-separated list of modules or simply "all"

mask = string: a space-separated list of output categories or simply
"everything"
```

NOTES

```
"feedback" alone will print a list of the available module choices
```

PYMOL API

```
cmd.feedback\(string action,string module,string mask\)
```

EXAMPLES

```
feedback enable, all , debugging
feedback disable, selector, warnings actions
feedback enable, main, blather
```

api: pymol.feedingback.feedback

---

fetch

DESCRIPTION

```
"fetch" downloads a file from the internet \(if possible\)
```

USAGE

```
fetch code \[, name \[, state \[, finish \[, discrete \[, multiplex
    \[, zoom \[, type \[, async \[, path \]\]\]\]\]\]\]\]\]\]
```

ARGUMENTS

```
code = a single PDB identifier or a list of identifiers. Supports
5-letter codes for fetching single chains \(like 1a0OA\).

name = the object name into which the file should be loaded.

state = the state number into which the file should loaded.

type = str: cif, pdb, pdb1, 2fofc, fofc, emd, cid, sid \{default: cif
\(default was "pdb" up to 1.7.6\)\}

async\_ = 0/1: download in the background and do not block the PyMOL
command line \{default: 0 -- changed in PyMOL 2.3\}
```

PYMOL API

```
cmd.fetch\(string code, string name, int state, init finish,
          int discrete, int multiplex, int zoom, string type,
          int async, string path, string file, int quiet\)
```

NOTES

```
When running in interactive mode, the fetch command loads
structures asyncronously by default, meaning that the next command
may get executed before the structures have been loaded.  If you
need synchronous behavior in order to insure that all structures
are loaded before the next command is executed, please provide the
optional argument "async=0".

Fetch requires a direct connection to the internet and thus may
not work behind certain types of network firewalls.
```

api: pymol.importing.fetch

---

fit

DESCRIPTION

```
"fit" superimposes the model in the first selection on to the model
in the second selection. Only matching atoms in both selections will
be used for the fit.
```

USAGE

```
fit mobile, target \[, mobile\_state \[, target\_state \[, quiet
    \[, matchmaker \[, cutoff \[, cycles \[, object \]\]\]\]\]\]\]
```

ARGUMENTS

```
mobile = string: atom selection

target = string: atom selection

mobile\_state = integer: object state \{default=0, all states\)

target\_state = integer: object state \{default=0, all states\)

matchmaker = integer: how to match atom pairs \{default: 0\}
    -1:  assume that atoms are stored in the identical order
    0/1: match based on all atom identifiers \(segi,chain,resn,resi,name,alt\)
    2:   match based on ID
    3:   match based on rank
    4:   match based on index \(same as -1 \?\)

cutoff = float: outlier rejection cutoff \(only if cycles>0\) \{default: 2.0\}

cycles = integer: number of cycles in outlier rejection refinement \{default:
0\}

object = string: name of alignment object to create \{default: None\}
```

EXAMPLES

```
fit protA, protB
```

NOTES

```
Since atoms are matched based on all of their identifiers
\(including segment and chain identifiers\), this command is only
helpful when comparing very similar structures.
```

SEE ALSO

```
[align](https://pymol.org/pymol-command-ref.html#align), [super]
(https://pymol.org/pymol-command-ref.html#super), [pair\_fit]
(https://pymol.org/pymol-command-ref.html#pair_fit), [rms]
(https://pymol.org/pymol-command-ref.html#rms), [rms\_cur]
(https://pymol.org/pymol-command-ref.html#rms_cur), [intra\_fit]
(https://pymol.org/pymol-command-ref.html#intra_fit), [intra\_rms]
(https://pymol.org/pymol-command-ref.html#intra_rms), [intra\_rms\_cur]
(https://pymol.org/pymol-command-ref.html#intra_rms_cur)
```

api: pymol.fitting.fit

fix_chemistry

DESCRIPTION

```
"fix chemistry" is an unsupported feature.
```

api: pymol.editing.fix_chemistry

flag

DESCRIPTION

```
"flag" sets the indicated flag for atoms in the selection and
 clears the indicated flag for atoms not in the selection.
```

USAGE

```
flag flag, selection \[, action \]
```

ARGUMENTS

```
action = reset: \{default\} set flag for atoms in selection and clear it for all
others

action = set: set the flag for atoms in selection, leaving other atoms unchanged

action = clear: clear the flag for selected atoms, leaving other atoms unchanged
```

EXAMPLES

```
flag free, \(resi 45 x; 6\)
```

NOTES

```
This is primarily useful for passing selection information into
Chempy models, which have a 32 bit attribute "flag" which holds
this information.

If the 'auto\_indicate\_flags' setting is true, then PyMOL will automatically
create a selection called "indicate" which contains all atoms with that flag
after applying the command.

SPECIAL FLAGS

\* Flags 0-5 are reserved for molecular modeling

    focus      0 = Atoms of Interest \(i.e. a ligand in an active site\) \\
    free       1 = Free Atoms \(free to move subject to a force-field\) \\
    restrain   2 = Restrained Atoms \(typically harmonically contrained\) \\
```

```
    fix        3 = Fixed Atoms \(no movement allowed\) \\
    exclude    4 = Atoms which should not be part of any simulation
    study      5

\* Flags 6-7 are for protein and nucleic acid classification

\* Flags 8-15 are free for end users to manipulate

\* Flags 16-21 are reserved for external GUIs and linked applications

\* Flags 22-23 are for temporary use only \(flag 23 used for coverage
  tracking when assigning parameters in chempy.champ.assign\)

\* Flags 24-31 are reserved for PyMOL internal usage

    exfoliate 24 = Remove surface from atoms when surfacing \\
    ignore    25 = Ignore atoms altogether when surfacing \\
    no\_smooth 26 = Do not smooth atom position
```

PYMOL API

```
cmd.flag\(int flag, string selection, string action="reset",
         int indicate=0\)
```

api: pymol.editing.flag

---

fnab

DESCRIPTION

```
Builds a nucleotide acid from sequence
```

USAGE

```
fnab input \[, name \[, type \[, form \[, dbl\_helix \]\]\]\]
```

ARGUMENTS

```
input = str: Sequence as an array of one letter codes

name = str: Name of the object to create \{default: obj\}

mode = str: "DNA" or "RNA"

form = str: "A" or "B"

dbl\_helix = bool \(0/1\): flag for using double helix in DNA
```

EXAMPLE

```
fnab ATGCGATAC
fnab ATGCGATAC, name=myDNA, mode=DNA, form=B, dbl\_helix=1
fnab AAUUUUCCG, mode=RNA
```

api: pymol.editor.fnab

focal_blur

DESCRIPTION

```
Creates fancy figures by introducing a focal blur to the image.
The object at the origin will be in focus.
```

USAGE

```
focal\_blur \[ aperture \[, samples \[, ray \[, filename \]\]\]\]
```

ARGUMENTS

```
aperture = float: aperture angle in degrees \{default: 2.0\}

samples = int: number of images for averaging \{default: 10\}

ray = 0/1: \{default: 0\}

filename = str: write image to file \{default: temporary\}
```

AUTHORS

```
Jarl Underhaug, Jason Vertrees and Thomas Holder
```

EXAMPLES

```
focal\_blur 3.0, 50
```

api: pymol.experimenting.focal_blur

fork

DESCRIPTION

```
"spawn" launches a Python script in a new thread which will run
concurrently with the PyMOL interpreter. It can be run in its own
namespace \(like a Python module, default\), a local name space, or
in the global namespace.
```

USAGE

```
spawn file \[, namespace \]
```

NOTES

```
The default namespace for spawn is "module".

The best way to spawn processes at startup is to use the -l option
\(see "help launching"\).
```

SEE ALSO

```
[run](https://pymol.org/pymol-command-ref.html#run)
```

api: pymol.parsing.spawn

---

forward

DESCRIPTION

```
"forward" moves the movie one frame forward.
```

USAGE

```
forward
```

PYMOL API

```
cmd.forward\(\)
```

SEE ALSO

```
[mset](https://pymol.org/pymol-command-ref.html#mset), [backward]
(https://pymol.org/pymol-command-ref.html#backward), [rewind]
(https://pymol.org/pymol-command-ref.html#rewind)
```

api: pymol.moving.forward

---

fragment

DESCRIPTION

```
"fragment" retrieves a 3D structure from the fragment library,
which is currently pretty meager \(just amino acids\).
```

USAGE

```
fragment name
```

api: pymol.creating.fragment

---

frame

DESCRIPTION

```
"frame" sets the viewer to the indicated movie frame.
```

USAGE

```
frame frame
```

ARGUMENTS

```
frame = integer: frame number to display
```

EXAMPLE

```
frame 10
```

PYMOL API

```
cmd.frame\( int frame\_number \)
```

NOTES

```
Frame numbers are 1-based.
```

SEE ALSO

```
[count\_states](https://pymol.org/pymol-command-ref.html#count_states)
```

api: pymol.moving.frame

---

full_screen

DESCRIPTION

```
"full\_screen" enables or disables full screen mode.
```

USAGE

```
full\_screen \[toggle\]
```

EXAMPLES

```
full\_screen
full\_screen on
full\_screen off
```

NOTES

```
This does not work correctly on all platforms.  If you encounter
trouble, try using the maximize button on the viewer window
instead.
```

api: pmg_qt.pymol_qt_gui.full_screen

---

fuse

DESCRIPTION

```
"fuse" joins two objects into one by forming a bond.  A copy of
the object containing the first atom is moved so as to form an
approximately resonable bond with the second, and that copy is
then merged with the first object.
```

USAGE

```
fuse \[ selection1 \[, selection2 \[, mode \[, recolor \[, move \]\]\]\]\]
```

ARGUMENTS

```
selection1 = str: single atom selection \(will be copied to object 2\)

selection2 = str: single atom selection

mode = int: \{default: 0\}
  3: don't move and don't create a bond, just combine into single object

recolor = bool: recolor C atoms to match target \{default: 1\}

move = bool: \{default: 1\}
```

NOTES

```
Each selection must include a single atom in each object.
The atoms can both be hydrogens, in which case they are
eliminated, or they can both be non-hydrogens, in which
case a bond is formed between the two atoms.
```

SEE ALSO

```
[bond](https://pymol.org/pymol-command-ref.html#bond), [unbond]
(https://pymol.org/pymol-command-ref.html#unbond), [attach]
(https://pymol.org/pymol-command-ref.html#attach), [replace]
(https://pymol.org/pymol-command-ref.html#replace), [fuse]
(https://pymol.org/pymol-command-ref.html#fuse), [remove\_picked]
(https://pymol.org/pymol-command-ref.html#remove_picked)
```

api: pymol.editing.fuse

---

get

DESCRIPTION

```
"get" prints out the current value of a setting.
```

USAGE

```
get name \[, selection \[, state \]\]
```

EXAMPLE

```
get line\_width
```

ARGUMENTS

```
name = string: setting name

selection = string: object name \(selections not yet supported\)

state = integer: state number
```

NOTES

```
"get" currently only works with global, per-object, and per-state
settings.  Atom level settings get be queried with "iterate" \(e.g.
iterate all, print s.line\_width\)
```

PYMOL API

```
cmd.get\(string name, string object, int state, int quiet\)
```

SEE ALSO

```
[set](https://pymol.org/pymol-command-ref.html#set), [set\_bond]
(https://pymol.org/pymol-command-ref.html#set_bond), [get\_bond]
(https://pymol.org/pymol-command-ref.html#get_bond)
```

api: pymol.setting.get

---

get_angle

DESCRIPTION

```
"get\_angle" returns the angle between three atoms.  By default, the
coordinates used are from the current state, however an alternate
state identifier can be provided.
```

USAGE

```
get\_angle atom1, atom2, atom3, \[,state \]
```

EXAMPLES

```
get\_angle 4/n,4/c,4/ca
get\_angle 4/n,4/c,4/ca,state=4
```

PYMOL API

```
cmd.get\_angle\(atom1="pk1",atom2="pk2",atom3="pk3",state=-1\)
```

api: pymol.querying.get_angle

---

get_area

DESCRIPTION

```
Get the surface area of an selection. Depends on the "dot\_solvent"
setting. With "dot\_solvent=off" \(default\) it calculates the solvent
excluded surface area, else the surface accessible surface.
```

USAGE

```
get\_area \[ selection \[, state \[, load\_b \]\]\]
```

ARGUMENTS

```
load\_b = bool: store per-atom surface area in b-factors \{default: 0\}
```

SEE ALSO

```
"dot\_solvent" setting, "dots" representation \([show](https://pymol.org/pymol-
command-ref.html#show) dots\)
```

api: pymol.querying.get_area

---

get_bond

DESCRIPTION

```
"get\_bond" gets per-bond settings for all bonds which exist
between two selections of atoms.
```

USAGE

```
get\_bond name, selection1 \[, selection2 \]
```

ARGUMENTS

```
name = string: name of the setting

selection1 = string: first set of atoms

selection2 = string: seconds set of atoms \{default: \(selection1\)\}
```

EXAMPLE

```
get\_bond stick\_transparency, \*/n+c+ca+o
```

NOTES

```
The following per-bond settings are currently implemented.  Others
may seem to be recognized but will currently have no effect when
set at the per-bond level.

\* valence
\* line\_width
\* line\_color
\* stick\_radius
\* stick\_color
\* stick\_transparency
```

PYMOL API

```
cmd.get\_bond \( string name,
                string selection1,
                string selection2,
                int state, int updates, quiet=1\)
```

api: pymol.setting.get_bond

---

get_chains

DESCRIPTION

```
Print the list of chain identifiers in the given selection.
```

USAGE

```
get\_chains \[ selection \[, state \]\]
```

ARGUMENTS

```
selection = str: atom selection \{default: all\}

state = int: CURRENTLY IGNORED
```

api: pymol.querying.get_chains

---

get_dihedral

DESCRIPTION

```
"get\_dihedral" returns the dihedral angle between four atoms.  By
default, the coordinates used are from the current state, however
an alternate state identifier can be provided.

By convention, positive dihedral angles are right-handed
\(looking down the atom2-atom3 axis\).
```

USAGE

```
get\_dihedral atom1, atom2, atom3, atom4 \[,state \]
```

EXAMPLES

```
get\_dihedral 4/n,4/c,4/ca,4/cb
get\_dihedral 4/n,4/c,4/ca,4/cb,state=4
```

PYMOL API

```
cmd.get\_dihedral\(atom1,atom2,atom3,atom4,state=-1\)
```

api: pymol.querying.get_dihedral

---

get_distance

DESCRIPTION

```
"get\_distance" returns the distance between two atoms.  By default, the
coordinates used are from the current state, however an alternate
state identifier can be provided.
```

USAGE

```
get\_distance atom1, atom2, \[,state \]
```

EXAMPLES

```
get\_distance 4/n,4/c
get\_distance 4/n,4/c,state=4
```

PYMOL API

```
cmd.get\_distance\(atom1="pk1",atom2="pk2",state=-1\)
```

api: pymol.querying.get_distance

---

get_extent

DESCRIPTION

```
"get\_extent" returns the minimum and maximum XYZ coordinates of a
selection as an array:
 \[ \[ min-X , min-Y , min-Z \],\[ max-X, max-Y , max-Z \]\]
```

PYMOL API

```
cmd.get\_extent\(string selection="\(all\)", state=0 \)
```

api: pymol.querying.get_extent

---

get_position

DESCRIPTION

```
"get\_position" returns the 3D coordinates of the center of the
viewer window.
```

api: pymol.querying.get_position

---

get_property

DESCRIPTION

```
Get an object-level property
```

ARGUMENTS

```
propname = string: Name of the property

name = string: Name of a single object

state = int: Object state, 0 for all states, -1 for current state
\{default: 0\}
```

api: pymol.properties.get_property

---

get_property_list

DESCRIPTION

```
Get all properties for an object \(for a particular state\) as a list
```

ARGUMENTS

```
object = string: Name of a single object

state = int: Object state, 0 for all states, -1 for current state
\{default: 0\}
```

api: pymol.properties.get_property_list

---

get_renderer

DESCRIPTION

```
Prints OpenGL renderer information.
```

api: pymol.querying.get_renderer

---

get_sasa_relative

DESCRIPTION

```
Calculates the relative per-residue solvent accessible surface area
and optionally labels and colors residues. The value is relative to
full exposure of the residue, calculated by removing all other
residues except its two next neighbors, if present.

Loads a value beteween 0.0 \(fully buried\) and 1.0 \(fully exposed\)
into the b-factor property, available in "iterate", "alter" and
"label" as "b".
```

USAGE

```
get\_sasa\_relative \[ selection \[, state \[, vis \[, var \]\]\]\]
```

ARGUMENTS

```
selection = str: atom selection \{default: all\}

state = int: object state \{default: 1\}

vis = 0/1: show labels and do color by exposure \{default: \!quiet\}

var = str: name of property to assign \{default: b\}

quiet = 0/1: print results to log window

outfile = str: filename, write to file instead of log window \{default: \}
```

EXAMPLE

```
fetch 1ubq, async=0
get\_sasa\_relative polymer
```

PYTHON API

```
cmd.get\_sasa\_relative\(...\) -> dict
```

SEE ALSO

```
[get\_area](https://pymol.org/pymol-command-ref.html#get_area) with "load\_b=1"
argument.
```

api: pymol.util.get_sasa_relative

---

get_symmetry

DESCRIPTION

```
"get\_symmetry" can be used to obtain the crystal
and spacegroup parameters for a molecule or map.
```

USAGE

```
get\_symmetry object-name-or-selection
```

PYMOL API

```
cmd.get\_symmetry\(string selection, int state, int quiet\)
```

api: pymol.querying.get_symmetry

---

get_title

DESCRIPTION

```
"get\_title" retrieves a text string to the state of a particular
object which will be displayed when the state is active.
```

USAGE

```
set\_title object, state
```

PYMOL API

```
cmd.set\_title\(string object, int state, string text\)
```

api: pymol.querying.get_title

---

get_type

DESCRIPTION

```
"get\_type" returns a string describing the named object or
 selection or the string "nonexistent" if the name in unknown.
```

PYMOL API

```
cmd.get\_type\(string object-name\)
```

NOTES

```
Possible return values are

"object:molecule"
"object:map"
"object:mesh"
"object:slice"
"object:surface"
"object:measurement"
"object:cgo"
"object:group"
"object:volume"
"selection"
```

SEE ALSO

```
get\_names
```

api: pymol.querying.get_type

---

get_version

DESCRIPTION

```
"get\_version" returns a tuple of length six containing text,
floating point, and integer representations of the current PyMOL
version number, build date as unix timestamp, GIT SHA and SVN
code revision so far available.
```

PYMOL API

```
cmd.get\_version\(int quiet\)
```

api: pymol.querying.get_version

---

get_view

DESCRIPTION

```
"get\_view" returns and optionally prints out the current view
information in a format which can be embedded into a command
script and can be used in subsequent calls to "set\_view".

If a log file is currently open, get\_view will not write the view
matrix to the screen unless the "output" parameter is 2.
```

USAGE

```
get\_view \[output\]
```

ARGUMENTS

```
output = 0: output matrix to screen

output = 1: do not Output matrix to screen

output = 2: force output to screen even if log file is open

output = 3: return formatted string instead of a list
```

NOTES

```
Contents of the view matrix:

\* 0  -  8: column-major 3x3 matrix which rotates model space to camera space

\* 9  - 11: origin of rotation relative to camera \(in camera space\)

\* 12 - 14: origin of rotation \(in model space\)

\* 15: front plane distance from the camera

\* 16: rear plane distance from the camera

\* 17: orthoscopic flag \(+/-\) and field of view \(if abs\(value\) > 1\)

The camera always looks down -Z with its +X left and its +Y down.

Therefore, in the default view, model +X is to the observer's
right, +Y is upward, and +Z points toward the observer.
```

PYMOL API

```
cmd.get\_view\(output=1, quiet=1\)
```

SEE ALSO

```
[set\_view](https://pymol.org/pymol-command-ref.html#set_view)
```

api: pymol.viewing.get_view

get_viewport

DESCRIPTION

```
"get\_viewport" returns and optionally prints out the screen viewport size

If a log file is currently open, get\_viewport will not write the view
matrix to the screen unless the "output" parameter is 2.
```

USAGE

```
get\_viewport \[output\]
```

ARGUMENTS

```
output = 0: output matrix to screen

output = 1: do not Output matrix to screen

output = 2: force output to screen even if log file is open

output = 3: return formatted string instead of a list
```

PYMOL API

```
cmd.get\_viewport\(output=1, quiet=1\)
```

api: pymol.viewing.get_viewport

---

gradient

DESCRIPTION

```
"gradient" creates a gradient object from a map object.
```

USAGE

```
gradient name, map \[, minimum \[, maximum \[, selection \[, buffer \[, state
    \[, carve \[, source\_state \[, quiet \]\]\]\]\]\]\]\]
```

ARGUMENTS

```
map = the name of the map object to use.

minimum, maximum = minimum and maximum levels \(default: full map range\)

selection = an atom selection about which to display the mesh with
    an additional "buffer" \(if provided\).
```

SEE ALSO

```
[load](https://pymol.org/pymol-command-ref.html#load), [isomesh]
(https://pymol.org/pymol-command-ref.html#isomesh)
```

api: pymol.creating.gradient

---

group

DESCRIPTION

```
"group" creates or updates a group object: a container for
organizing objects into a hierarchy.
```

USAGE

```
group name \[, members \[, action \]\]
```

ARGUMENTS

```
name = string: name of the group

members = string: space-separated list of objects to include in
          the group

action = add, remove, open, close, toggle, auto, empty,
         purge, excise
```

ACTIONS

```
add:     add members to group
remove:  remove members from group \(members will be ungrouped\)
empty:   remove all members from group
purge:   remove all members from group and delete them
excise:  remove all members from group and delete group
open:    expand group display in object menu panel
close:   collapse group display in object menu panel
toggle:  toggle group display in object menu panel
auto:    add or toggle
ungroup: DEPRECATED, use ungroup command
```

EXAMPLE

```
group kinases, 1oky 1pkg 1t46 1uwh 1z5m
group kinases, open
group kinases, close
```

NOTES

```
Group objects can typically be used as arguments to commands.  In
such cases, the command should be applied to all members of the
group.  If the group is used as a selection, then all atoms in all
objects in the group should be included in the selection.

When a group objects is open, objects can be added or removed from
the group by right-clicking and dragging in the control panel.
```

SEE ALSO

```
[ungroup](https://pymol.org/pymol-command-ref.html#ungroup), [order]
(https://pymol.org/pymol-command-ref.html#order), "group\_auto\_mode" setting
```

api: pymol.creating.group

---

h_add

DESCRIPTION

```
"h\_add" adds hydrogens onto a molecule based on current valences.
```

USAGE

```
h\_add \[ selection \[, state \]\]
```

ARGUMENTS

```
selection = string \{default: \(all\)\}

state = int \{default: 0 \(all states\)\}
```

NOTES

```
Because PDB files do not normally contain bond valences for
ligands and other nonstandard components, it may be necessary to
manually correct ligand conformations before adding hydrogens.
```

SEE ALSO

```
[h\_fill](https://pymol.org/pymol-command-ref.html#h_fill)
```

api: pymol.editing.h_add

---

h_fill

DESCRIPTION

```
"h\_fill" removes and replaces hydrogens on the atom or bond picked
for editing.
```

USAGE

```
h\_fill
```

NOTES

```
This is useful for fixing hydrogens after changing bond valences.
```

PYMOL API

```
cmd.h\_fill\(\)
```

SEE ALSO

```
[edit](https://pymol.org/pymol-command-ref.html#edit), [cycle\_valence]
(https://pymol.org/pymol-command-ref.html#cycle_valence), [h\_add]
(https://pymol.org/pymol-command-ref.html#h_add)
```

api: pymol.editing.h_fill

---

h_fix

DESCRIPTION

```
"h\_fix" is an unsupported command that may have something to do
with repositioning hydrogen atoms.
```

api: pymol.editing.h_fix

---

help

DESCRIPTION

```
"help" prints out the online help for a given command.
```

USAGE

```
help command
```

api: pymol.helping.help

---

help_setting

DESCRIPTION

```
Print documentation for a setting.
```

USAGE

```
help\_setting name
```

api: pymol.helping.help_setting

---

hide

DESCRIPTION

```
"hide" turns off atom and bond representations.
```

USAGE

```
hide \[ representation \[, selection \]\]
```

ARGUMENTS

```
representation = lines, spheres, mesh, ribbon, cartoon,
   sticks, dots, surface, labels, extent, nonbonded, nb\_spheres,
   slice, extent, slice, dashes, angles, dihedrals, cgo, cell, callback,
   or everything

selection = string: a selection-expression or name-pattern
```

EXAMPLES

```
hide lines, all
hide ribbon
```

PYMOL API

```
cmd.hide\(string representation, string selection\)
```

SEE ALSO

```
[show](https://pymol.org/pymol-command-ref.html#show), [enable]
(https://pymol.org/pymol-command-ref.html#enable), [disable]
(https://pymol.org/pymol-command-ref.html#disable)
```

api: pymol.viewing.hide

---

id_atom

DESCRIPTION

```
"id\_atom" returns the original source id of a single atom, or
raises and exception if the atom does not exist or if the selection
corresponds to multiple atoms.
```

PYMOL API

```
list = cmd.id\_atom\(string selection\)
```

api: pymol.querying.id_atom

---

identify

DESCRIPTION

```
"identify" returns a list of atom IDs corresponding to the ID code
of atoms in the selection.
```

PYMOL API

```
list = cmd.identify\(string selection="\(all\)",int mode=0\)
```

NOTES

```
mode 0: only return a list of identifiers \(default\)
mode 1: return a list of tuples of the object name and the identifier
```

api: pymol.querying.identify

---

index

DESCRIPTION

```
"index" returns a list of tuples corresponding to the
object name and index of the atoms in the selection.
```

PYMOL API

```
list = cmd.index\(string selection="\(all\)"\)
```

NOTE

Atom indices are fragile and will change as atoms are added or deleted. Whenever possible, use integral atom identifiers instead of indices.

api: pymol.querying.index

---

indicate

DESCRIPTION

```
"indicate" shows a visual representation of an atom selection.
```

USAGE

```
indicate \(selection\)
```

PYMOL API

```
cmd.count\(string selection\)
```

api: pymol.selecting.indicate

---

intra_fit

DESCRIPTION

```
    "intra\_fit" fits all states of an object to an atom selection
    in the specified state.  It returns the rms values to python
    as an array.
```

USAGE

```
    intra\_fit selection \[, state\]
```

ARGUMENTS

```
    selection = string: atoms to fit

    state = integer: target state
```

PYMOL API

```
    cmd.intra\_fit\( string selection, int state \)
```

EXAMPLES

```
    intra\_fit \( name CA \)
```

PYTHON EXAMPLE

```
    from pymol import cmd
    rms = cmd.intra\_fit\("\(name CA\)",1\)
```

SEE ALSO

```
    [fit](https://pymol.org/pymol-command-ref.html#fit), [rms]
(https://pymol.org/pymol-command-ref.html#rms), [rms\_cur]
(https://pymol.org/pymol-command-ref.html#rms_cur), [intra\_rms]
(https://pymol.org/pymol-command-ref.html#intra_rms), [intra\_rms\_cur]
(https://pymol.org/pymol-command-ref.html#intra_rms_cur), [pair\_fit]
(https://pymol.org/pymol-command-ref.html#pair_fit)
```

api: pymol.fitting.intra_fit

---

intra_rms

DESCRIPTION

```
    "intra\_rms" calculates rms fit values for all states of an object
    over an atom selection relative to the indicated state.
    Coordinates are left unchanged.  The rms values are returned as a
    python array.
```

EXAMPLE

```
    from pymol import cmd
    rms = cmd.intra\_rms\("\(name CA\)",1\)
    print rms
```

PYMOL API

```
    cmd.intra\_rms\(string selection, int state\)
```

SEE ALSO

```
    [fit](https://pymol.org/pymol-command-ref.html#fit), [rms]
(https://pymol.org/pymol-command-ref.html#rms), [rms\_cur]
(https://pymol.org/pymol-command-ref.html#rms_cur), [intra\_fit]
(https://pymol.org/pymol-command-ref.html#intra_fit), [intra\_rms\_cur]
(https://pymol.org/pymol-command-ref.html#intra_rms_cur), [pair\_fit]
(https://pymol.org/pymol-command-ref.html#pair_fit)
```

api: pymol.fitting.intra_rms

---

intra_rms_cur

DESCRIPTION

```
    "intra\_rms\_cur" calculates rms values for all states of an object
    over an atom selection relative to the indicated state without
    performing any fitting.  The rms values are returned
    as a python array.
```

PYMOL API

```
    cmd.intra\_rms\_cur\( string selection, int state\)
```

PYTHON EXAMPLE

```
    from pymol import cmd
    rms = cmd.intra\_rms\_cur\("\(name CA\)",1\)
```

SEE ALSO

```
    [fit](https://pymol.org/pymol-command-ref.html#fit), [rms]
(https://pymol.org/pymol-command-ref.html#rms), [rms\_cur]
(https://pymol.org/pymol-command-ref.html#rms_cur), [intra\_fit]
(https://pymol.org/pymol-command-ref.html#intra_fit), [intra\_rms]
(https://pymol.org/pymol-command-ref.html#intra_rms), [pair\_fit]
(https://pymol.org/pymol-command-ref.html#pair_fit)
```

api: pymol.fitting.intra_rms_cur

---

invert

DESCRIPTION

```
"invert" inverts the stereo-chemistry of atom \(pk1\), holding attached atoms
\(pk2\) and \(pk3\) immobile.
```

USAGE

```
invert
```

NOTES

```
The invert function is usually bound to CTRL-E in Editing Mode.
```

PYMOL API

```
cmd.invert\( \)
```

api: pymol.editing.invert

---

isodot

DESCRIPTION

```
"isodot" creates a dot isosurface object from a map object.
```

USAGE

```
isodot name, map \[, level \[, selection \[, buffer \[, state
    \[, carve \[, source\_state \[, quiet \]\]\]\]\]\]\]
```

ARGUMENTS

```
map = the name of the map object to use.

level = the contour level.

selection = an atom selection about which to display the mesh with
    an additional "buffer" \(if provided\).
```

NOTES

```
If the dot isosurface object already exists, then the new dots will
be appended onto the object as a new state.
```

SEE ALSO

```
[load](https://pymol.org/pymol-command-ref.html#load), [isomesh]
(https://pymol.org/pymol-command-ref.html#isomesh)
```

api: pymol.creating.isodot

---

isolevel

DESCRIPTION

```
"isolevel" changes the contour level of a isodot, isosurface, or isomesh object.
```

USAGE

```
isolevel name, level, state
```

api: pymol.creating.isolevel

---

isomesh

DESCRIPTION

```
"isomesh" creates a mesh isosurface object from a map object.
```

USAGE

```
isomesh name, map, level \[, selection \[, buffer \[, state \[, carve \]\]\]\]
```

ARGUMENTS

```
name = the name for the new mesh isosurface object.

map = the name of the map object to use for computing the mesh.

level = the contour level.

selection = an atom selection about which to display the mesh with
    an additional "buffer" \(if provided\).

state = the state into which the object should be loaded \(default=1\)
    \(set state=0 to append new mesh as a new state\)

carve = a radius about each atom in the selection for which to
    include density. If "carve" is not provided, then the whole
    brick is displayed.
```

NOTES

```
If the mesh object already exists, then the new mesh will be
appended onto the object as a new state \(unless you indicate a state\).

state > 0: specific state
state = 0: all states
state = -1: current state

source\_state > 0: specific state
source\_state = 0: include all states starting with 0
source\_state = -1: current state
source\_state = -2: last state in map
```

SEE ALSO

```
[isodot](https://pymol.org/pymol-command-ref.html#isodot), [load]
(https://pymol.org/pymol-command-ref.html#load)
```

api: pymol.creating.isomesh

---

isosurface

DESCRIPTION

```
"isosurface" creates a new surface object from a map object.
```

USAGE

```
isosurface name, map, level \[, selection \[, buffer \[, state \[, carve
\]\]\]\]
```

ARGUMENTS

```
name = the name for the new mesh isosurface object.

map = the name of the map object to use for computing the mesh.

level = the contour level.

selection = an atom selection about which to display the mesh with
    an additional "buffer" \(if provided\).

state = the state into which the object should be loaded \(default=1\)
    \(set state=0 to append new surface as a new state\)

carve = a radius about each atom in the selection for which to
    include density. If "carve= not provided, then the whole
    brick is displayed.
```

NOTES

```
If the surface object already exists, then the new surface will be
appended onto the object as a new state \(unless you indicate a state\).
```

SEE ALSO

```
[isodot](https://pymol.org/pymol-command-ref.html#isodot), [isomesh]
(https://pymol.org/pymol-command-ref.html#isomesh), [load]
(https://pymol.org/pymol-command-ref.html#load)
```

api: pymol.creating.isosurface

---

iterate

DESCRIPTION

```
"iterate" iterates over an expression within a temporary namespace
for each atom.
```

USAGE

```
iterate selection, expression
```

EXAMPLES

```
stored.net\_charge = 0
iterate all, stored.net\_charge = stored.net\_charge + partial\_charge
print stored.net\_charge

stored.names = \[\]
iterate all, stored.names.append\(name\)
print stored.names
```

NOTES

```
Unlike with the "alter" command, atomic properties cannot be
altered.  For this reason, "iterate" is more efficient than
"alter".
```

SEE ALSO

```
[iterate\_state](https://pymol.org/pymol-command-ref.html#iterate_state), [alter]
(https://pymol.org/pymol-command-ref.html#alter), [alter\_state]
(https://pymol.org/pymol-command-ref.html#alter_state)
```

api: pymol.editing.iterate

---

iterate_state

DESCRIPTION

```
"iterate\_state" is to "alter\_state" as "iterate" is to "alter"
```

USAGE

```
iterate\_state state, selection, expression
```

EXAMPLES

```
stored.sum\_x = 0.0
iterate\_state 1, all, stored.sum\_x = stored.sum\_x + x
print stored.sum\_x
```

SEE ALSO

```
[iterate](https://pymol.org/pymol-command-ref.html#iterate), [alter]
(https://pymol.org/pymol-command-ref.html#alter), [alter\_state]
(https://pymol.org/pymol-command-ref.html#alter_state)
```

api: pymol.editing.iterate_state

---

join_states

DESCRIPTION

```
The reverse of split\_states. Create a multi-state object from a
selection which spans multiple objects.
```

ARGUMENTS

```
name = string: name of object to create or modify

selection = string: atoms to include in the new object

mode = int: how to match states \{default: 2\}
  0: Create discrete object, input objects can be \(totally\) different
  1: Assume identical topology \(same number of atoms and matching atom
     identifiers\) in all input objects
  2: Assume matching atom identifiers in all input objects, but also
     check for missing atoms and only include atoms that are present
     in all input objects
  3: match atoms by sequence alignment, slowest but most robust option
```

EXAMPLE

```
fragment ala
fragment his
join\_states multi, \(ala|his\), mode=0
```

api: pymol.creating.join_states

---

label

DESCRIPTION

```
"label" labels one or more atoms in a selection by evaluating an
Python expression referencing properties for each atom.
```

USAGE

```
label \[ selection \[, expression \]\]
```

ARGUMENTS

```
selection = string: a selection-expression

expression = string: a Python expression that can be converted to a string
```

EXAMPLES

```
label chain A, chain
label name CA,"\%s-\%s" \% \(resn,resi\)
label resi 200,"\%1.3f" \% partial\_charge
```

NOTES

```
The symbols defined in the label name space for each atom are:

    name, resi, resn, resv, chain, segi, model, alt, q, b, type,
    index, rank, ID, ss, vdw, elec\_radius, label, elem, geom,
    flags, color, cartoon, valence, formal\_charge, partial\_charge,
    numeric\_type, text\_type, stereo

All strings in the expression must be explicitly quoted.

This operation typically takes several seconds per thousand atoms
labelled.

To clear labels, simply omit the expression or set it to ''.
```

api: pymol.viewing.label

---

load

DESCRIPTION

```
"load" can by used to read molecules, crystallographic maps and
other volumetric data, PyMOL sessions, and some other types of
content.
```

USAGE

```
load filename \[, object \[, state \[, format \[, finish \[, discrete \[, quiet
        \[, multiplex \[, zoom \[, partial \[, mimic \[, object\_props
        \[, atom\_props \]\]\]\]\]\]\]\]\]\]\]\]\]
```

ARGUMENTS

```
filename = string: file path or URL

object = string: name of the object \{default: filename prefix\}

state = integer: number of the state into which
the content should be loaded, or 0 for append \{default:0\}

format = pdb, ccp4, etc. \{default: use file extension\}\): format of
data file

partial = for session files, the partial flag loads the session while
preserving what is currently loaded in PyMOL.  If it is set to 2,
then the view that is in the session gets set to the current view

object\_props = argument specifies whether properties should be loaded
into PyMOL.  If it is a space-delimited list, then only these properties
will get loaded. If it is a '\*', then all properties get loaded.
If an empty string, then no properties get loaded. \{default: use
load\_object\_props\_default setting\}

atom\_props = argument specifies whether atom properties should
be loaded into PyMOL.  If it is a space-delimited list, then only these
properties will get loaded. If it is a '\*', then all properties get loaded.
If an empty string, then no properties get loaded. \{default: use
load\_atom\_props\_default setting\}
```

EXAMPLES

```
load 1dn2.pdb

load file001.pdb, ligand

load http://delsci.com/sample.pdb

load file002.mae, ligand, object\_props=\*
```

NOTES

```
The file extension is used to determine the format unless the
format is provided explicitly.

If an object name is specified, then the file is loaded into that
object.  Otherwise, an object is created with the same name as the
file prefix.

If a state value is not specified, then the content is appended
after the last existing state \(if any\).

Supported molecular file formats include: pdb, mol, mol2, sdf,
```

```
xyz, and others.

Supported map formats include: xplor, ccp4, phi, mtz, and others.

All supported file formats are covered in the reference
documentation under File Formats.
```

PYMOL API

```
cmd.load\(string filename, string object-name, integer state,
         string format, int finish, int discrete, int quiet,
         int multiplex, int zoom, int partial, string object\_props,
         string atom\_props\)
```

SEE ALSO

```
[save](https://pymol.org/pymol-command-ref.html#save), [load\_traj]
(https://pymol.org/pymol-command-ref.html#load_traj), [fetch]
(https://pymol.org/pymol-command-ref.html#fetch)
```

api: pymol.importing.load

---

load_embedded

DESCRIPTION

```
"load\_embedded" loads content previously defined in the current
PyMOL command script using the "embed" command.
```

USAGE

```
load\_embedded \[ key \[, name \[, state \[, finish \[, discrete \[, quiet
\]\]\]\]\]\]\]
```

EXAMPLE

```
embed wats, pdb
HETATM    1  O   WAT     1       2.573  -1.034  -1.721
HETATM    2  H1  WAT     1       2.493  -1.949  -1.992
HETATM    3  H2  WAT     1       2.160  -0.537  -2.427
HETATM    4  O   WAT     2       0.705   0.744   0.160
HETATM    5  H1  WAT     2      -0.071   0.264   0.450
HETATM    6  H2  WAT     2       1.356   0.064  -0.014
embed end

load\_embedded wats
```

NOTES

```
This approach only works with text data files.
```

api: pymol.importing.load_embedded

load_mtz

DESCRIPTION

```
Load a MTZ file as two map objects \(fofc, 2fofc\) or if amplitudes and
phases column names are given, as one map object.
```

USAGE

```
load\_mtz filename \[, prefix \[, amplitudes, phases \[, weights
    \[, reso\_low \[, reso\_high \]\]\]\]\]
```

ARGUMENTS

```
filename = str: filename

prefix = str: object name or prefix \{default: filename without extension\}

amplitudes = str: amplitudes column name, guess if blank \{default: \}

phases = str: phases column name, required if amplitudes are given \{default: \}

weights = str: weights column name, optional \{default: None\}

reso\_low = float: minimum resolution \{default: 0, read from file\}

reso\_high = float: maximum resolution \{default: 0, read from file\}
```

api: pymol.importing.load_mtz

load_png

DESCRIPTION

```
"load\_png" loads and displays a PNG file from disk.
```

USAGE

```
load\_png filename
```

NOTES

```
If the displayed image is too big for the window, it will be
reduced 2-fold repeatedly until it fits.
```

api: pymol.viewing.load_png

load_traj

DESCRIPTION

```
"load\_traj" reads trajectory files.

Most of the trajectory formats listed here are supported:
http://www.ks.uiuc.edu/Research/vmd/plugins/molfile/
```

USAGE

```
load\_traj filename \[,object \[,state \[,format \[,interval \[,average \]
                        \[,start \[,stop \[,max \[,selection \[,image \[,shift
                        \]\]\]\]\]\]\]\]\]
```

ARGUMENTS

```
filename = str: path to trajectory file

object = str: name of the molecular object where the trajectory should be
appended as states \{default: guess from filename or last object in list\}

state = int: first object state to populate, or 0 to append after
last state \{default: 0\}

format = str: file format \{default: guess from extension\}

interval = int: interval to take frames from file \{default: 1\}

average = int: \? \(trj only, possibly broken\)

start = int: first frame to load from file \{default: 1\}

stop = int: last frame to load from file, or -1 to load all \{default: -1\}

max = int: maximum number of states to load, or 0 to load all \{default: 0\}

selection = str: atom selection to only load a subset of coordinates
\{default: all\}

image = 0/1: residue-based period image transformation \(trj only\)

shift = float-3: offset for image transformation \{default: \(0,0,0\}

plugin = str: name of VMD plugin to use \{default: guess from magic string
of from format\}
```

PYMOL API

```
cmd.load\_traj\(filename,object='',state=0,format='',interval=1,
              average=1,start=1,stop=-1,max=-1,selection='all',image=1,
              shift="\[0.0,0.0,0.0\]"\)
```

NOTES

```
You must first load a corresponding topology file before attempting
to load a trajectory file.

PyMOL does not know how to wrap the truncated octahedron used by Amber
You will need to use the "ptraj" program first to do this.

The average option is not a running average.  To perform this type of
average, use the "smooth" command after loading the trajectory file.
```

SEE ALSO

```
[load](https://pymol.org/pymol-command-ref.html#load)
```

api: pymol.importing.load_traj

---

loadall

DESCRIPTION

```
Load all files matching given globbing pattern
```

USAGE

```
loadall pattern \[, group \]
```

EXAMPLE

```
loadall \*.pdb
```

api: pymol.importing.loadall

---

log

DESCRIPTION

```
"log" writes a command to the log file \(if one is open\).

\`text\` and/or \`alt\_text\` must include the terminating line feed.
```

ARGUMENTS

```
text = str: PyMOL command \(optional if alt\_text is given\)
alt\_text = str: Python expression \(optional\)
```

SEE ALSO

```
[log\_open](https://pymol.org/pymol-command-ref.html#log_open), [log\_close]
(https://pymol.org/pymol-command-ref.html#log_close)
```

api: pymol.commanding.log

log_close

DESCRIPTION

```
"log\_close" closes the current log file \(if one is open\).
```

USAGE

```
log\_close
```

SEE ALSO

```
[log](https://pymol.org/pymol-command-ref.html#log), [log\_open]
(https://pymol.org/pymol-command-ref.html#log_open)
```

api: pymol.commanding.log_close

log_open

DESCRIPTION

```
"log\_open" opens a log file for writing.
```

USAGE

```
log\_open \[ filename \[, mode \]\]
```

ARGUMENTS

```
filename = str: file to write to \(.pml or .py\) \{default: log.pml\}

mode = w/a: "w" to open an empty log file, "a" to append \{default: w\}
```

SEE ALSO

```
[log](https://pymol.org/pymol-command-ref.html#log), [log\_close]
(https://pymol.org/pymol-command-ref.html#log_close)
```

api: pymol.commanding.log_open

ls

DESCRIPTION

```
List contents of the current working directory.
```

USAGE

```
ls \[pattern\]
dir \[pattern\]
```

EXAMPLES

```
ls
ls \*.pml
```

SEE ALSO

```
[cd](https://pymol.org/pymol-command-ref.html#cd), [pwd](https://pymol.org/pymol-
command-ref.html#pwd), [system](https://pymol.org/pymol-command-ref.html#system)
```

api: pymol.externing.ls

---

madd

DESCRIPTION

```
"madd" extends the existing movie specification using the same
syntax as mset.
```

SEE ALSO

```
[mset](https://pymol.org/pymol-command-ref.html#mset), [mdo]
(https://pymol.org/pymol-command-ref.html#mdo), [mplay](https://pymol.org/pymol-
command-ref.html#mplay), [mclear](https://pymol.org/pymol-command-
ref.html#mclear)
```

api: pymol.moving.madd

---

map_double

DESCRIPTION

```
"map\_double" resamples a map at twice the current resolution.
```

NOTES

```
 The amount of memory required to store the map will increase
 eight-fold.
```

USAGE

```
map\_double map\_name, state
```

api: pymol.editing.map_double

---

map_halve

DESCRIPTION

```
"map\_halve" resamples a map at half the current resolution.
```

USAGE

```
map\_halve map\_name, state
```

NOTES

```
The amount of memory required to store the map will decrease
eight-fold.
```

SEE ALSO

```
[map\_double](https://pymol.org/pymol-command-ref.html#map_double)
```

api: pymol.editing.map_halve

---

map_new

DESCRIPTION

```
"map\_new" creates a map object using one of the built-in map
generation routines.  This command not yet fully supported.
```

USAGE

```
map\_new name \[, type \[, grid \[, selection \[, buffer \[, box \[, state
\]\]\]\]\]\]
```

ARGUMENTS

```
name = string: name of the map object to create or modify

type = vdw, gaussian, gaussian\_max, coulomb, coulomb\_neutral, coulomb\_local

grid = float: grid spacing

selection = string: atoms about which to generate the map

buffer = float: cutoff

state > 0: use the indicated state

state = 0: use all states independently with independent extents

state = -1: use current global state

state = -2: use effective object state\(s\)
```

```
state = -3: use all states in one map

state = -4: use all states independent states by with a unified extent
```

NOTES

```
This command can be used to create low-resolution surfaces of
protein structures.
```

api: pymol.creating.map_new

---

map_set

DESCRIPTION

```
"map\_set" provides a number of common operations on and between maps.
```

USAGE

```
map\_set name, operator, operands, target\_state, source\_state

operator may be "minimum, maximum, average, sum, or difference"
```

EXAMPLES

```
map my\_sum, add, map1 map2 map3
map my\_avg, average, map1 map2 map3
```

NOTES

```
source\_state = 0 means all states
target\_state = -1 means current state

experimental
```

SEE ALSO

```
[map\_new](https://pymol.org/pymol-command-ref.html#map_new)
```

api: pymol.editing.map_set

---

map_set_border

DESCRIPTION

```
"map\_set\_border" is a function \(reqd by PDA\) which allows you to set the
level on the edge points of a map
```

USAGE

```
map\_set\_border name, level
```

NOTES

```
unsupported.
```

SEE ALSO

```
[load](https://pymol.org/pymol-command-ref.html#load)
```

api: pymol.editing.map_set_border

---

map_trim

DESCRIPTION

```
"map\_trim" is an unsupported command that may have something to do with
reducing the extent of a map to cover just a single selection of atoms.
```

api: pymol.editing.map_trim

---

mappend

DESCRIPTION

```
"mappend" associates additional command line operations with a
particular movie frame.  These "generalized movie commands" will
be executed every time the numbered frame is played.
```

USAGE

```
mappend frame: command
```

ARGUMENTS

```
frame = integer: the frame to modify

command = literal command-line text
```

EXAMPLE

```
mappend 1: hide everything; show sticks
mappend 60: hide sticks; show spheres
mappend 120: hide spheres; show surface
```

NOTES

```
The "mset" command must first be used to define the movie before
"mdo" statements will have any effect.  Redefinition of the movie
clears any existing movie commands specified with mdo or mappend.
```

SEE ALSO

```
[mset](https://pymol.org/pymol-command-ref.html#mset), [madd]
(https://pymol.org/pymol-command-ref.html#madd), [mdo](https://pymol.org/pymol-
command-ref.html#mdo), [mplay](https://pymol.org/pymol-command-ref.html#mplay),
[mstop](https://pymol.org/pymol-command-ref.html#mstop)
```

api: pymol.moving.mappend

---

mask

DESCRIPTION

```
"mask" makes it impossible to select the indicated atoms using the
mouse.  This is useful when you are working with one molecule in
front of another and wish to avoid accidentally selecting atoms in
the background.
```

USAGE

```
mask \(selection\)
```

PYMOL API

```
cmd.mask\( string selection="\(all\)" \)
```

SEE ALSO

```
[unmask](https://pymol.org/pymol-command-ref.html#unmask), [protect]
(https://pymol.org/pymol-command-ref.html#protect), [deprotect]
(https://pymol.org/pymol-command-ref.html#deprotect), [mouse]
(https://pymol.org/pymol-command-ref.html#mouse)
```

api: pymol.controlling.mask

---

matrix_copy

DESCRIPTION

```
"matrix\_copy" copies a transformation matrix from one object to
another.
```

USAGE

```
matrix\_copy source\_name, target\_name
```

NOTES

```
This command is often used after a protein structure alignment to
bring other related objects into the same frame of reference.
```

SEE ALSO

```
[matrix\_reset](https://pymol.org/pymol-command-ref.html#matrix_reset), [align]
(https://pymol.org/pymol-command-ref.html#align), [fit](https://pymol.org/pymol-
command-ref.html#fit), [pair\_fit](https://pymol.org/pymol-command-
ref.html#pair_fit)
```

api: pymol.editing.matrix_copy

---

matrix_reset

DESCRIPTION

```
"matrix\_reset" resets the transformation for an object.
```

USAGE

```
matrix\_reset name \[, state \[, mode \]\]
```

ARGUMENTS

```
name = str: object name

state = int: object state \{default: 1\}

mode = int: \{defualt: -1 = matrix\_mode or 0\}
  0: transformation was applied to coordinates
  1: reset TTT matrix \(movie transformation\)
  2: reset state matrix
```

SEE ALSO

```
[matrix\_copy](https://pymol.org/pymol-command-ref.html#matrix_copy), [align]
(https://pymol.org/pymol-command-ref.html#align), [super]
(https://pymol.org/pymol-command-ref.html#super), [fit](https://pymol.org/pymol-
command-ref.html#fit), [pair\_fit](https://pymol.org/pymol-command-
ref.html#pair_fit)
```

api: pymol.editing.matrix_reset

---

mclear

DESCRIPTION

```
"mclear" clears the movie frame image cache.
```

USAGE

```
mclear
```

PYMOL API

```
cmd.mclear\(\)
```

api: pymol.moving.mclear

---

mcopy

DESCRIPTION

```
"mcopy" copies key frames and movie commands

Usage like "mmove".
```

SEE ALSO

```
[mmove](https://pymol.org/pymol-command-ref.html#mmove), [mdelete]
(https://pymol.org/pymol-command-ref.html#mdelete), [minsert]
(https://pymol.org/pymol-command-ref.html#minsert)
```

api: pymol.moving.mcopy

---

mdelete

DESCRIPTION

```
"mdelete" removes frames from camera view and object motions.
```

ARGUMENTS

```
count = int: number of frames to delete, or -1 to delete all the way
to the end \{default: -1\}

frame = int: first frame to delete, or 0 for current frame \{default: 0\}
```

EXAMPLE

```
# delete frames 81 to 90
mdelete 10, 81
```

SEE ALSO

```
[minsert](https://pymol.org/pymol-command-ref.html#minsert), [mmove]
(https://pymol.org/pymol-command-ref.html#mmove)
```

api: pymol.moving.mdelete

mdo

DESCRIPTION

```
"mdo" defines \(or redefines\) the command-line operations
associated with a particular movie frame.  These "generalized
movie commands" will be executed every time the numbered frame is
played.
```

USAGE

```
mdo frame: command
```

PYMOL API

```
cmd.mdo\( int frame, string command \)
```

EXAMPLE

```
// Creates a single frame movie involving a rotation about X and Y

load test.pdb
mset 1
mdo 1, turn x,5; turn y,5;
mplay
```

NOTES

These commands are usually created by a PyMOL utility program (such as movie.rock). Command can actually contain several commands separated by semicolons ';'

```
The "mset" command must first be used to define the movie before
"mdo" statements will have any effect.  Redefinition of the movie
clears any existing mdo statements.
```

SEE ALSO

```
[mset](https://pymol.org/pymol-command-ref.html#mset), [mplay]
(https://pymol.org/pymol-command-ref.html#mplay), [mstop]
(https://pymol.org/pymol-command-ref.html#mstop)
```

api: pymol.moving.mdo

---

mdump

DESCRIPTION

```
"mdump" dumps the current set of movie commands as text output.
```

USAGE

```
mdump
```

SEE ALSO

```
[mplay](https://pymol.org/pymol-command-ref.html#mplay), [mset]
(https://pymol.org/pymol-command-ref.html#mset), [mdo](https://pymol.org/pymol-
command-ref.html#mdo), [mclear](https://pymol.org/pymol-command-ref.html#mclear),
[mmatrix](https://pymol.org/pymol-command-ref.html#mmatrix)
```

api: pymol.moving.mdump

---

mem

DESCRIPTION

```
"mem" Dumps current memory state to standard output. This is a
debugging feature, not an official part of the API.
```

api: pymol.experimenting.mem

---

meter_reset

DESCRIPTION

```
"meter\_reset" resets the frames per secound counter.
```

USAGE

```
meter\_reset
```

api: pymol.viewing.meter_reset

---

middle

DESCRIPTION

```
"middle" goes to the middle of the movie.
```

USAGE

```
middle
```

PYMOL API

```
cmd.middle\(\)
```

api: pymol.moving.middle

---

minsert

DESCRIPTION

```
"minsert" adds frames into camera view and object motions.
```

ARGUMENTS

```
count = int: number of frames to insert

frame = int: insert before "frame" if frame > 0, otherwise insert before
the current frame \{default: 0\}
```

SEE ALSO

```
[mdelete](https://pymol.org/pymol-command-ref.html#mdelete), [mmove]
(https://pymol.org/pymol-command-ref.html#mmove), [madd](https://pymol.org/pymol-
command-ref.html#madd)
```

api: pymol.moving.minsert

---

mmatrix

DESCRIPTION

```
"mmatrix" sets up a matrix to be used for the first frame of the movie.
```

USAGE

```
mmatrix action
```

ARGUMENTS

```
action = clear, store, or recall
```

NOTES

```
This command ensures that the movie always starts from the same
camera view.

"mmatrix" should not be used when controlling the camera using
"mview".
```

PYMOL API

```
cmd.mmatrix\( string action \)
```

EXAMPLES

```
mmatrix store
```

api: pymol.moving.mmatrix

mmove

DESCRIPTION

```
"mmove" moves key frames and movie commands
```

ARGUMENTS

```
target = int: frame to move to

source = int: frame to move from, 0 for current frame \{default: 0\}

count = int: number of frames to move
```

SEE ALSO

```
[mcopy](https://pymol.org/pymol-command-ref.html#mcopy), [mdelete]
(https://pymol.org/pymol-command-ref.html#mdelete), [minsert]
(https://pymol.org/pymol-command-ref.html#minsert)
```

api: pymol.moving.mmove

morph

DESCRIPTION

```
Creates an interpolated trajectory between two or multiple conformations.
If the two input objects are not the same, match them based on sequence
alignment.

This command supports two methods: rigimol and linear. RigiMOL is an
incentive feature and only available to official PyMOL sponsors. Linear
morphing is quick and robust but likely to produce distorted intermediates.
```

ARGUMENTS

```
name = string: name of object to create

sele1 = string: atom selection of first conformation

sele2 = string: atom selection of second conformation \{default: \<sele1>\}

state1 = int: sele1 state \{default: 1\}. If state1=0 and sele1 has N
states, create N morphings between all consecutive states and back from
state N to 1 \(so the morph will have N\*steps states\). If state2=0, create
N-1 morphings and stop at last state.

state2 = int: sele2 state \{default: 2 if sele1=sele2, else 1\}

refinement = int: number of sculpting refinement cycles to clean
```

```
distorted intermediates \{default: 3\}

steps = int: number of states for sele2 object \{default: 30\}

method = string: rigimol or linear \{default: rigimol\}
```

EXAMPLE

```
fetch 1akeA 4akeA, async=0
align 1akeA, 4akeA
morph mout, 1akeA, 4akeA
```

api: pymol.morphing.morph

---

mouse

DESCRIPTION

```
"mouse" cycles through the mouse modes defined in the current
mouse configuration ring.
```

USAGE

```
mouse
```

api: pymol.controlling.mouse

---

move

DESCRIPTION

```
"move" translates the camera about one of the three primary axes.
```

USAGE

```
move axis, distance
```

EXAMPLES

```
move x, 3
move y, -1
```

PYMOL API

```
cmd.move\(string axis, float distance\)
```

SEE ALSO

```
[turn](https://pymol.org/pymol-command-ref.html#turn), [rotate]
(https://pymol.org/pymol-command-ref.html#rotate), [translate]
(https://pymol.org/pymol-command-ref.html#translate), [zoom]
(https://pymol.org/pymol-command-ref.html#zoom), [center]
(https://pymol.org/pymol-command-ref.html#center), [clip]
(https://pymol.org/pymol-command-ref.html#clip)
```

api: pymol.viewing.move

---

movie.load

UNDOCUMENTED

api: pymol.movie.load

---

movie.nutate

UNDOCUMENTED

api: pymol.movie.nutate

---

movie.pause

UNDOCUMENTED

api: pymol.movie.pause

---

movie.produce

DESCRIPTION

```
Export a movie to an MPEG file.

Requires FREEMOL.
```

ARGUMENTS

```
filename = str: filename of MPEG file to produce

mode = draw or ray: \{default: check "ray\_trace\_frames" setting\}

first = int: first frame to export \{default: 1\}

last = int: last frame to export \{default: last frame of movie\}

preserve = 0 or 1: don't delete temporary files \{default: 0\}

quality = 0-100: encoding quality \{default: 90 \(movie\_quality setting\)\}
```

api: pymol.movie.produce

---

movie.rock

UNDOCUMENTED

api: pymol.movie.rock

movie.roll

UNDOCUMENTED

api: pymol.movie.roll

movie.screw

UNDOCUMENTED

api: pymol.movie.screw

movie.sweep

UNDOCUMENTED

api: pymol.movie.sweep

movie.tdroll

AUTHOR

```
Byron DeLaBarre
```

USAGE

```
movie.tdroll\(rangx,rangey,rangez,skip=1,mset=0\)

rangex/y/z = rotation range on respective axis
enter 0 for no rotation.

skip is angle increment in each frame

Use skip to reduce final movie size or to speed up rotation.
```

EXAMPLE

```
movie.tdroll 360,360,360,5
```

api: pymol.movie.tdroll

movie.zoom

UNDOCUMENTED

api: pymol.movie.zoom

mplay

DESCRIPTION

```
"mplay" starts the movie.
```

USAGE

```
mplay
```

PYMOL API

```
cmd.mplay\(\)
```

SEE ALSO

```
[mstop](https://pymol.org/pymol-command-ref.html#mstop), [mset]
(https://pymol.org/pymol-command-ref.html#mset), [mdo](https://pymol.org/pymol-
command-ref.html#mdo), [mclear](https://pymol.org/pymol-command-ref.html#mclear),
[mmatrix](https://pymol.org/pymol-command-ref.html#mmatrix)
```

api: pymol.moving.mplay

mpng

DESCRIPTION

```
"mpng" writes movie frames as a series of numbered png files.
```

USAGE

```
mpng prefix \[, first \[, last \[, preserve \[, modal \[, mode \[, quiet
    \[, width \[, height \]\]\]\]\]\]\]\]
```

ARGUMENTS

```
prefix = string: filename prefix for saved images -- output files
will be numbered and end in ".png"

first = integer: starting frame \{default: 0 \(first frame\)\}

last = integer: last frame \{default: 0 \(last frame\)\}

preserve = 0/1: Only write non-existing files \{default: 0\}

modal = integer: will frames be rendered with a modal draw loop

mode = int: 2=ray, 1=draw, 0=normal \{default: -1, check
ray\_trace\_frames or draw\_frames\}
```

```
width = int: width in pixels \{default: current viewport\}

height = int: height in pixels \{default: current viewport\}
```

NOTES

```
If the "ray\_trace\_frames" variable is non-zero, then the frames
will be ray-traced.  Note that this can take many hours for a long
movie with complex content displayed.

Also, be sure to avoid setting "cache\_frames" when rendering a
long movie to avoid running out of memory.

Arguments "first" and "last" can be used to specify an inclusive
interval over which to render frames.  Thus, you can write a smart
Python program that will automatically distribute rendering over a
cluster of workstations.  If these options are left at zero, then
the entire movie will be rendered.
```

PYMOL API

```
cmd.mpng\(string prefix, int first, int last\)
```

SEE ALSO

```
[png](https://pymol.org/pymol-command-ref.html#png), [save]
(https://pymol.org/pymol-command-ref.html#save)
```

api: pymol.moving.mpng

---

mse2met

DESCRIPTION

```
Mutate selenomethionine to methionine
```

api: pymol.editing.mse2met

---

mset

DESCRIPTION

```
"mset" sets up a relationship between molecular states and movie
frames.  This makes it possible to control which states are shown
in which frame.
```

USAGE

```
mset specification \[ ,frame \]
```

PYMOL API

```
cmd.mset\( string specification \[, int frame\] \)
```

EXAMPLES

```
# simplest case, one state -> one frame

mset 1

# ten frames, all corresponding to state 1

mset 1 x10

# the first thirty frames are state 1
# the next 15 frames pass through states 1-15
# the next 30 frames are of state 15
# the next 15 frames iterate back to state 1

mset 1 x30 1 -15 15 x30 15 -1
```

SEE ALSO

```
[madd](https://pymol.org/pymol-command-ref.html#madd), [mdo]
(https://pymol.org/pymol-command-ref.html#mdo), [mplay](https://pymol.org/pymol-
command-ref.html#mplay), [mclear](https://pymol.org/pymol-command-
ref.html#mclear)
```

api: pymol.moving.mset

---

mstop

DESCRIPTION

```
"mstop" stops playing of the movie.
```

USAGE

```
mstop
```

SEE ALSO

```
[mplay](https://pymol.org/pymol-command-ref.html#mplay), [mset]
(https://pymol.org/pymol-command-ref.html#mset), [mdo](https://pymol.org/pymol-
command-ref.html#mdo), [mclear](https://pymol.org/pymol-command-ref.html#mclear),
[mmatrix](https://pymol.org/pymol-command-ref.html#mmatrix)
```

api: pymol.moving.mstop

---

mtoggle

DESCRIPTION

```
"mtoggle" toggles playing of the movie.
```

api: pymol.moving.mtoggle

---

multifilesave

DESCRIPTION

```
For a selection that spans multiple molecular objects and/or states,
save each object and/or state to a separate file. Takes a filename
argument with placeholders:

\{name\}  : object name
\{state\} : state number
\{title\} : state title
\{num\}   : file number
\{\}      : object name \(first\) or state \(second\)
```

EXAMPLES

```
multifilesave /tmp/\{name\}.pdb
multifilesave /tmp/\{name\}-\{state\}.cif, state=0
multifilesave /tmp/\{\}-\{\}.cif, state=0
multifilesave /tmp/\{\}-\{title\}.sdf, state=0
```

api: pymol.exporting.multifilesave

---

multisave

DESCRIPTION

```
"multisave" will save a multi-entry PDB file.

Every object in the given selection \(pattern\) will have a HEADER and a
CRYST \(if symmetry is defined\) record, and is terminated with END.
Loading such a multi-entry PDB file into PyMOL will load each entry
as a separate object.

This behavior is different to the "save" command, where a multi-object
selection is written "flat" to a PDB file, without HEADER or CRYST
records.
```

ARGUMENTS

```
filename = string: file path to be written

pattern = str: atom selection \(before 1.8.4: object name pattern\)

state = int: object state \(-1=current, 0=all\) \{default: -1\}

append = 0/1: append to existing file \{default: 0\}

format = str: file format \{default: guess from extension, or 'pdb'\}
```

api: pymol.exporting.multisave

---

mview

DESCRIPTION

```
"mview" stores camera and object matrices for use in movie
interpolation.
```

USAGE

```
mview \[action \[, first \[, last \[, power \[, bias \[, simple
   \[, linear \[, object \[, wrap \[, hand \[, window \[, cycles \[,scene
   \[, cut \[, quiet \]\]\]\]\]\]\]\]\]\]\]\]\]\]\]\]\]\]\]
```

ARGUMENTS

```
action = str: one of store, clear, reset, purge, interpolate,
uninterpolate, reinterpolate, toggle, toggle\_interp, smooth
\{default: store\}

first = int: frame number or 0 for current frame \{default: 0\}

power = float: slow down animation at keyframe \(0.0\) or not \(1.0\)
\{default: 0.0\}

object = str: name of object for object keyframes, or empty for
global \(camera\) keyframes \{default: \}

scene = str: name of scene to store scene with key frame \{default: \}

cut = float 0.0-1.0: scene switch moment \(0.0: beginning of transition,
1.0: end of transition\) \{default: 0.5\}

auto = -1/0/1: if freeze=0, then auto reinterpolate after store, clear,
or toggle \{default: -1 = use movie\_auto\_interpolate\}

state = int: if > 0, then store object state \{default: 0\}

freeze = 0/1: never auto reinterpolate \{default: 0\}
```

SEE ALSO

```
[mplay](https://pymol.org/pymol-command-ref.html#mplay), [mset]
(https://pymol.org/pymol-command-ref.html#mset), [mdo](https://pymol.org/pymol-
command-ref.html#mdo), [mclear](https://pymol.org/pymol-command-ref.html#mclear),
[mmatrix](https://pymol.org/pymol-command-ref.html#mmatrix)
```

api: pymol.moving.mview

---

order

DESCRIPTION

```
"order" changes the ordering of names in the control panel.
```

USAGE

```
order names, sort, location
```

ARGUMENTS

```
names = string: a space-separated list of names

sort = yes or no \{default: no\}

location = top, current, or bottom \{default: current\}
```

EXAMPLES

```
order 1dn2 1fgh 1rnd       # sets the order of these three objects
order \*,yes                # sorts all names
order 1dn2\_\*, yes          # sorts all names beginning with 1dn2\_
order 1frg, location=top   # puts 1frg at the top of the list
```

PYMOL API

```
cmd.order\(string names, string sort, string location\)
```

NOTES

```
"order" can also be used to reorder objects within a group.
```

SEE ALSO

```
[set\_name](https://pymol.org/pymol-command-ref.html#set_name), [group]
(https://pymol.org/pymol-command-ref.html#group)
```

api: pymol.controlling.order

---

orient

DESCRIPTION

```
"orient" aligns the principal components of the atoms in the
selection with the XYZ axes.
```

USAGE

```
orient \[ selection \[, state \[, animate \]\]\]
```

ARGUMENTS

```
selection = a selection-expression or name-pattern \{default: \(all\)\}

state = 0: use all coordinate states \{default\}

state = -1: uses only coordinates for the current state

state > 0: uses coordinates for a specific state
```

EXAMPLES

```
orient organic
```

NOTES

```
The function is similar to the orient command in X-PLOR.
```

PYMOL API

```
cmd.orient\(string object-or-selection, int state, float animate\)
```

SEE ALSO

```
[zoom](https://pymol.org/pymol-command-ref.html#zoom), [origin]
(https://pymol.org/pymol-command-ref.html#origin), [reset]
(https://pymol.org/pymol-command-ref.html#reset)
```

api: pymol.viewing.orient

---

origin

DESCRIPTION

```
"origin" sets the center of rotation about a selection.  If an
object name is specified, it can be used to set the center of
rotation for the object \(for use in animation and editing\).
```

USAGE

```
origin \[ selection \[, object \[,position, \[, state \]\]\]\]
```

ARGUMENTS

```
selection = string: selection-expression or name-list \{default: \(all\)\}

state = 0 \(default\) use all coordinate states

state = -1 use only coordinates for the current state

state > 0  use coordinates for a specific state
```

EXAMPLES

```
origin chain A

origin position=\[1.0,2.0,3.0\]
```

PYMOL API

```
cmd.origin\(string object-or-selection\)
```

SEE ALSO

```
[zoom](https://pymol.org/pymol-command-ref.html#zoom), [orient]
(https://pymol.org/pymol-command-ref.html#orient), [reset]
(https://pymol.org/pymol-command-ref.html#reset)
```

api: pymol.viewing.origin

---

overlap

DESCRIPTION

```
"overlap" is an unsupported command that sums up
\[\(VDWi + VDWj\) - distance\_ij\]/2 between pairs of
selected atoms.
```

PYMOL API

```
cmd.overlap\(string selection1, string selection2 \[, int state1=1, int state2=1,
float adjust=0.0\]\)
```

NOTES

```
It does not compute the volume overlap,
selections with more atoms will have a larger
sum.
```

api: pymol.querying.overlap

---

pair_fit

DESCRIPTION

```
"pair\_fit" fits matched sets of atom pairs between two objects.
```

USAGE

```
pair\_fit selection, selection, \[ selection, selection \[ ... \]\]
```

EXAMPLES

```
# superimpose protA residues 10-25 and 33-46 to protB residues 22-37 and 41-54:

pair\_fit protA/10-25+33-46/CA, protB/22-37+41-54/CA

# superimpose ligA atoms C1, C2, and C4 to ligB atoms C8, C4, and C10,
respectively:

pair\_fit ligA////C1, ligB////C8, ligA////C2, ligB////C4, ligA////C3,
ligB////C10
```

NOTES

```
So long as the atoms are stored in PyMOL with the same order
internally, you can provide just two selections.  Otherwise, you
may need to specify each pair of atoms separately, two by two, as
additional arguments to pair\_fit.

Script files are usually recommended when using this command.
```

SEE ALSO

```
[fit](https://pymol.org/pymol-command-ref.html#fit), [rms]
(https://pymol.org/pymol-command-ref.html#rms), [rms\_cur]
(https://pymol.org/pymol-command-ref.html#rms_cur), [intra\_fit]
(https://pymol.org/pymol-command-ref.html#intra_fit), [intra\_rms]
(https://pymol.org/pymol-command-ref.html#intra_rms), [intra\_rms\_cur]
(https://pymol.org/pymol-command-ref.html#intra_rms_cur)
```

api: pymol.fitting.pair_fit

---

phi_psi

DESCRIPTION

```
"phi\_psi" return the phi and psi angles for a protein atom
selection.
```

USAGE

api: pymol.querying.phi_psi

---

pi_interactions

DESCRIPTION

```
Find pi-pi and pi-cation interactions.

Identical to cmd.distance\(..., mode=5, label=0\)
```

SEE ALSO

```
[distance](https://pymol.org/pymol-command-ref.html#distance)
```

api: pymol.querying.pi_interactions

---

pip

DESCRIPTION

```
Experimental and limited pip support for installing additional Python
packages into PyMOL's bundled Python distribution. Type "pip help" for
more information.
```

EXAMPLE

```
pip install tkintertable
```

api: pymol.externing.pip

---

png

DESCRIPTION

```
"png" saves a PNG format image file of the current display.
```

USAGE

```
png filename \[, width \[, height \[, dpi \[, ray\]\]\]\]
```

ARGUMENTS

```
filename = string: file path to be written

width = integer or string: width in pixels \(without units\), inches \(in\)
or centimeters \(cm\). If unit suffix is given, dpi argument is required
as well. If only one of width or height is given, the aspect ratio of
the viewport is preserved. \{default: 0 \(current\)\}

height = integer or string: height \(see width\) \{default: 0 \(current\)\}

dpi = float: dots-per-inch \{default -1.0 \(unspecified\)\}

ray = 0 or 1: should ray be run first \{default: 0 \(no\)\}
```

EXAMPLES

```
png image.png
png image.png, dpi=300
png image.png, 10cm, dpi=300, ray=1
```

NOTES

```
PNG is the only image format supported by PyMOL.
```

SEE ALSO

```
[mpng](https://pymol.org/pymol-command-ref.html#mpng), [save]
(https://pymol.org/pymol-command-ref.html#save)
```

PYMOL API

```
cmd.png\(string filename, int width, int height, float dpi,
        int ray, int quiet\)
```

api: pymol.exporting.png

---

pop

DESCRIPTION

```
"pop" provides a mechanism of iterating through an atom selection
atom by atom, where each atom is sequentially assigned to the
named selection.
```

USAGE

```
pop name, source
```

EXAMPLE

```
select src, name CA

python
while cmd.pop\("tmp","src"\):
    cmd.zoom\("tmp",2, animate=1\)
    for a in range\(30\):
       cmd.refresh\(\)
       time.sleep\(0.05\)
python end
```

PYMOL API

```
cmd.deselect\(\)
```

api: pymol.selecting.pop

protect

DESCRIPTION

```
"protect" protects a set of atoms from tranformations performed
using the editing features.  This is most useful when you are
modifying an internal portion of a chain or cycle and do not wish
to affect the rest of the molecule.
```

USAGE

```
protect \(selection\)
```

PYMOL API

```
cmd.protect\(string selection\)
```

SEE ALSO

```
[deprotect](https://pymol.org/pymol-command-ref.html#deprotect), [mask]
(https://pymol.org/pymol-command-ref.html#mask), [unmask]
(https://pymol.org/pymol-command-ref.html#unmask), [mouse]
(https://pymol.org/pymol-command-ref.html#mouse), editing
```

api: pymol.editing.protect

pseudoatom

DESCRIPTION

```
"pseudoatom" adds a pseudoatom to a molecular object, and will
creating the molecular object if it does not yet exist.
```

USAGE

```
pseudoatom object \[, selection \[, name \[, resn \[, resi \[, chain
    \[, segi \[, elem \[, vdw \[, hetatm \[, b \[, q \[, color \[, label
    \[, pos \[, state \[, mode \[, quiet \]\]\]\]\]\]\]\]\]\]\]\]\]\]\]\]
```

NOTES

```
"pseudoatom" can be used for a wide variety of random tasks where
on must place an atom or a label in 3D space.
```

api: pymol.creating.pseudoatom

pwd

DESCRIPTION

```
Print current working directory.
```

USAGE

```
pwd
```

SEE ALSO

```
[cd](https://pymol.org/pymol-command-ref.html#cd), [ls](https://pymol.org/pymol-command-ref.html#ls), [system](https://pymol.org/pymol-command-ref.html#system)
```

api: pymol.externing.pwd

---

python

DESCRIPTION

```
"python" delimits a block of literal Python code embedded in a
PyMOL command script.
```

EXAMPLE

```
python

for a in range\(1,10\):
    b = 10 - a
    print a, b

python end
```

NOTES

```
Literal Python blocks avoid the annoying requirement of having to
use explicit line continuation markers for multi-line Python
commands embedded within Python scripts.
```

SEE ALSO

```
[abort](https://pymol.org/pymol-command-ref.html#abort), [embed]
(https://pymol.org/pymol-command-ref.html#embed), [skip](https://pymol.org/pymol-command-ref.html#skip)
```

api: pymol.helping.python

---

quit

DESCRIPTION

```
"quit" terminates the program.
```

USAGE

```
quit \[code\]
```

ARGUMENTS

```
code = int: exit the application with status "code" \{default: 0\}
```

PYMOL API

```
cmd.quit\(int code\)
```

api: pymol.commanding.quit

---

ramp_new

DESCRIPTION

```
"ramp\_new" creates a color ramp based on a map potential value or
based on proximity to a molecular object.
```

USAGE

```
ramp\_new name, map\_name \[, range \[, color \[, state \[, selection \[,
    beyond \[, within \[, sigma \[, zero \]\]\]\]\]\]\]\]
```

ARGUMENTS

```
name = string: name of the ramp object

map\_name = string: name of the map \(for potential\) or molecular
object \(for proximity\)

range = list: values corresponding to slots in the ramp

color = list: colors corresponding to slots in the ramp

state = integer: state identifier

selection = selection: for automatic ranging

beyond = number: with automatic ranging, are we excluding
values beyond a certain distance from the selection\?

within = number: with automatic ranging, are we only including
valuess within a certain distance from the selection\?

sigma = number: with automatic ranging, how many standard
deviations from the mean do we go\?

zero = integer: with automatic ranging, do we force the central
value to be zero\?
```

EXAMPLES

```
ramp\_new e\_pot\_color, e\_pot\_map, \[-10,0,10\], \[red,white,blue\]
```

NOTES

```
Color ramps are extremely powerful but complicated to use.

In the simplest case, they can be used to color representations
based on the potential values found in a map object at the
corresponding positions in space.

In another simple case, representations can be colored based on
proximity to a target.  Note that since ramp targets must
themselves be real objects \(not merely selections\), the "create"
command may be needed in order to generate an appropriate target.

In more complicated cases, they can be used to color
representations on one object based atoms found in another.

Ramps can operate recursively.  In other words, the output color
from one ramp can be used as the input color for another.  For
example, you could color by map potential within a certain
distance of the target object, beyond which, a uniform color is applied.
```

PYMOL API

```
def ramp\_new\(string name, string map\_name, list range, list color,
             int state, string selection, float beyond, float
             within, float sigma, int zero, int quiet\)
```

SEE ALSO

```
[ramp\_update](https://pymol.org/pymol-command-ref.html#ramp_update), [load]
(https://pymol.org/pymol-command-ref.html#load), [color](https://pymol.org/pymol-
command-ref.html#color), [create](https://pymol.org/pymol-command-
ref.html#create), slice, [gradient](https://pymol.org/pymol-command-
ref.html#gradient)
```

api: pymol.creating.ramp_new

---

ramp_update

DESCRIPTION

```
"ramp\_update" updates range and/or color of a color ramp.
```

USAGE

```
ramp\_update name \[, range \[, color \]\]
```

EXAMPLES

```
ramp\_new    e\_pot\_color, e\_pot\_map, \[-10,0,10\], \[red,white,blue\]
ramp\_update e\_pot\_color, range=\[-15,0,15\]
ramp\_update e\_pot\_color, color=\[green,white,orange\]
```

SEE ALSO

```
[ramp\_new](https://pymol.org/pymol-command-ref.html#ramp_new)
```

api: pymol.creating.ramp_update

---

ray

DESCRIPTION

```
"ray" creates a ray-traced image of the current frame. This
can take some time \(up to several minutes, depending on image
complexity\).
```

USAGE

```
ray \[width \[,height \[,antialias \[,angle \[,shift \[,renderer \[,quiet
    \[,async \]\]\]\]\]\]\]\]
```

ARGUMENTS

```
width = integer \{default: 0 \(current\)\}

height = integer \{default: 0 \(current\)\}

antialias = integer \{default: -1 \(use antialias setting\)\}

angle = float: y-axis rotation for stereo image generation
\{default: 0.0\}

shift = float: x-axis translation for stereo image generation
\{default: 0.0\}

renderer = -1, 0, 1, or 2: respectively, default, built-in,
pov-ray, or dry-run \{default: 0\}

async = 0 or 1: should rendering be done in a background thread\?
```

EXAMPLES

```
ray
ray 1024,768
ray renderer=2
```

NOTES

```
Default width and height are taken from the current viewpoint. If
one is specified but not the other, then the missing value is
scaled so as to preserve the current aspect ratio.

angle and shift can be used to generate matched stereo pairs

renderer = 1 uses PovRay.  This is Unix-only and you must have
    "povray" in your path.  It utilizes two two temporary files:
    "tmp\_pymol.pov" and "tmp\_pymol.png".

See "help faster" for optimization tips with the builtin renderer.
See "help povray" for how to use PovRay instead of PyMOL's
built-in ray-tracing engine.
```

PYMOL API

```
cmd.ray\(int width, int height, int antialias, float angle,
        float shift, int renderer, int quiet, int async\)
```

SEE ALSO

```
[draw](https://pymol.org/pymol-command-ref.html#draw), [png]
(https://pymol.org/pymol-command-ref.html#png), [save](https://pymol.org/pymol-
command-ref.html#save)
```

api: pymol.viewing.ray

---

rebond

DESCRIPTION

```
Discard all bonds and do distance based bonding.
```

ARGUMENTS

```
oname = str: object name

state = int: object state \{default: -1 \(current state\)\}
```

api: pymol.editing.rebond

---

rebuild

DESCRIPTION

```
"rebuild" forces PyMOL to recreate geometric objects in
case any of them have gone out of sync.
```

USAGE

```
rebuild \[selection \[, representation \]\]
```

ARGUMENTS

```
selection = string \{default: all\}

representation = string: \{default: everything\}
```

PYMOL API

```
cmd.rebuild\(string selection, string representation\)
```

SEE ALSO

```
[refresh](https://pymol.org/pymol-command-ref.html#refresh)
```

api: pymol.viewing.rebuild

---

recolor

DESCRIPTION

```
"recolor" forces reapplication of colors to existing objects.
```

USAGE

```
recolor \[selection \[, representation \]\]
```

ARGUMENTS

```
selection = string \{default: all\}

representation = string \{default: everything\}
```

NOTES

```
This command often needs to be executed after "set\_color" has been
used to redefine one or more existing colors.
```

PYMOL API

```
cmd.recolor\(string selection = 'all', string representation = 'everything'\)
```

SEE ALSO

```
[color](https://pymol.org/pymol-command-ref.html#color), [set\_color]
(https://pymol.org/pymol-command-ref.html#set_color)
```

api: pymol.viewing.recolor

---

redo

DESCRIPTION

```
"redo" reapplies the conformational change of the object currently
being edited.
```

USAGE

```
redo
```

SEE ALSO

```
[undo](https://pymol.org/pymol-command-ref.html#undo), push\_undo
```

api: pymol.editing.redo

---

reference

UNDOCUMENTED

api: pymol.editing.reference

---

refresh

DESCRIPTION

```
"refresh" causes the scene to be redrawn as soon as the operating
system allows it to be done.
```

USAGE

```
refresh
```

PYMOL API

```
cmd.refresh\(\)
```

SEE ALSO

```
[rebuild](https://pymol.org/pymol-command-ref.html#rebuild)
```

api: pymol.viewing.refresh

---

refresh_wizard

DESCRIPTION

```
"refresh\_wizard" is in unsupported internal command.
```

api: pymol.wizarding.refresh_wizard

---

reinitialize

DESCRIPTION

```
"reinitialize" reinitializes the program by deleting all objects
and restoring the default program settings.
```

USAGE

```
reinitialize
```

api: pymol.commanding.reinitialize

---

remove

DESCRIPTION

```
"remove" eleminates the atoms in a selection from their respective
molecular objects.
```

USAGE

```
remove selection
```

EXAMPLES

```
remove resi 124
```

PYMOL API

```
cmd.remove\( string selection \)
```

SEE ALSO

```
[delete](https://pymol.org/pymol-command-ref.html#delete)
```

api: pymol.editing.remove

---

remove_picked

DESCRIPTION

```
"remove\_picked" removes the atom or bond currently picked for
editing.
```

USAGE

```
remove\_picked \[ hydrogens \]
```

NOTES

```
This function is usually connected to the
DELETE key and "CTRL-D".

By default, attached hydrogens will also be deleted unless
hydrogen-flag is zero.
```

PYMOL API

```
cmd.remove\_picked\(integer hydrogens\)
```

SEE ALSO

```
[attach](https://pymol.org/pymol-command-ref.html#attach), [replace]
(https://pymol.org/pymol-command-ref.html#replace)
```

api: pymol.editing.remove_picked

---

rename

DESCRIPTION

```
"rename" creates new atom names which are unique within residues.
```

USAGE

```
rename selection \[, force \]
```

PYMOL API

```
cmd.rename\(string selection, int force \)
```

SEE ALSO

```
[alter](https://pymol.org/pymol-command-ref.html#alter)
```

api: pymol.editing.rename

---

replace

DESCRIPTION

```
"replace" replaces the picked atom with a new atom.
```

USAGE

```
replace element, geometry, valence \[, h\_fill \[, name \]\]
```

NOTES

```
Immature functionality. See code for details.
```

PYMOL API

```
cmd.replace\(string element, int geometry, int valence, int h\_fill,
            string name\)
```

SEE ALSO

```
[remove](https://pymol.org/pymol-command-ref.html#remove), [attach]
(https://pymol.org/pymol-command-ref.html#attach), [fuse]
(https://pymol.org/pymol-command-ref.html#fuse), [bond](https://pymol.org/pymol-
command-ref.html#bond), [unbond](https://pymol.org/pymol-command-
ref.html#unbond)
```

api: pymol.editing.replace

---

replace_wizard

DESCRIPTION

```
"replace\_wizard" is an unsupported internal command.
```

api: pymol.wizarding.replace_wizard

---

reset

DESCRIPTION

```
"reset" restores the rotation matrix to identity, sets the origin
to the center of mass \(approx.\) and zooms the window and clipping
planes to cover all objects.  Alternatively, it can reset object
matrices.
```

USAGE

```
reset \[ object \]
```

PYMOL API

```
cmd.reset\(\)
```

api: pymol.viewing.reset

---

resume

DESCRIPTION

```
"resume" executes a log file and opens it for recording of
additional commands.
```

USAGE

```
resume filename
```

SEE ALSO

```
[log](https://pymol.org/pymol-command-ref.html#log), [log\_close]
(https://pymol.org/pymol-command-ref.html#log_close)
```

api: pymol.commanding.resume

---

rewind

DESCRIPTION

```
"rewind" goes to the beginning of the movie.
```

USAGE

```
rewind
```

PYMOL API

```
cmd.rewind\(\)
```

api: pymol.moving.rewind

---

rms

DESCRIPTION

```
"rms" computes a RMS fit between two atom selections, but does not
tranform the models after performing the fit.
```

USAGE

```
rms \(selection\), \(target-selection\)
```

EXAMPLES

```
fit \( mutant and name CA \), \( wildtype and name CA \)
```

SEE ALSO

```
[fit](https://pymol.org/pymol-command-ref.html#fit), [rms\_cur]
(https://pymol.org/pymol-command-ref.html#rms_cur), [intra\_fit]
(https://pymol.org/pymol-command-ref.html#intra_fit), [intra\_rms]
(https://pymol.org/pymol-command-ref.html#intra_rms), [intra\_rms\_cur]
(https://pymol.org/pymol-command-ref.html#intra_rms_cur), [pair\_fit]
(https://pymol.org/pymol-command-ref.html#pair_fit)
```

api: pymol.fitting.rms

---

rms_cur

DESCRIPTION

```
"rms\_cur" computes the RMS difference between two atom
selections without performing any fitting.
```

USAGE

```
rms\_cur \(selection\), \(selection\)
```

SEE ALSO

```
[fit](https://pymol.org/pymol-command-ref.html#fit), [rms]
(https://pymol.org/pymol-command-ref.html#rms), [intra\_fit]
(https://pymol.org/pymol-command-ref.html#intra_fit), [intra\_rms]
(https://pymol.org/pymol-command-ref.html#intra_rms), [intra\_rms\_cur]
(https://pymol.org/pymol-command-ref.html#intra_rms_cur), [pair\_fit]
(https://pymol.org/pymol-command-ref.html#pair_fit)
```

api: pymol.fitting.rms_cur

---

rock

DESCRIPTION

```
"rock" toggles Y axis rocking.
```

USAGE

```
rock
```

PYMOL API

```
cmd.rock\(\)
```

api: pymol.viewing.rock

---

rotate

DESCRIPTION

```
"rotate" rotates the atomic coordinates of atoms in a selection
about an axis.  Alternatively, it modifies the matrix associated
with a particular object or object state.
```

USAGE

```
rotate axis, angle \[, selection \[, state \[, camera \[, object
    \[, origin \]\]\]\]\]
```

ARGUMENTS

```
axis = x, y, z, or float vector: axis about which to rotate

angle = float: degrees of rotation

selection = string: atoms whose coordinates should be modified \{default: all\}

state > 0: only the indicated state is modified

state = 0: all states are modified

state = -1: only the current state is modified \{default\}

camera = 0 or 1: is the axis specific in camera coordinates\? \{default: 1\}

object = string: object name \(only if rotating object matrix\) \{default:
None\}

origin = float vector: origin of rotateion \{default: None\}
```

EXAMPLES

```
rotate x, 45, pept

rotate \[1,1,1\], 10, chain A
```

NOTES

```
Behavior differs depending on whether or not the "object"
parameter is specified.

If object is None, then the atomic coordinates are modified
directly, and all representation geometries will need to be
regenerated to reflect the new atomic coordinates.

If object is set to an object name, then the selection field is
ignored and instead of translating the atomic coordinates, the
object matrix is modified.  This option is only intended for use
in animations and is not yet fully supported.
```

PYMOL API

```
cmd.rotate\(list-or-string axis, float angle, string selection,
           int state, int camera, string object\)
```

api: pymol.editing.rotate

---

run

DESCRIPTION

```
"run" executes an external Python script in a local name space,
the main Python namespace, the global PyMOL namespace, or in its
own namespace \(as a module\).
```

USAGE

```
run file \[, namespace \]
```

ARGUMENTS

```
file = string: a Python program, typically ending in .py or .pym.

namespace = local, global, module, main, or private \{default: global\}
```

NOTES

```
Due to an idiosyncracy in Pickle, you can not pickle objects
directly created at the main level in a script run as "module",
\(because the pickled object becomes dependent on that module\).
Workaround: delegate construction to an imported module.
```

SEE ALSO

```
[spawn](https://pymol.org/pymol-command-ref.html#spawn)
```

api: pymol.parsing.run

---

save

DESCRIPTION

```
"save" writes content to a file.
```

USAGE

```
save filename \[, selection \[, state \[, format \]\]\]
```

ARGUMENTS

```
filename = string: file path to be written

selection = string: atoms to save \{default: \(all\)\}

state = integer: state to save \{default: -1 \(current state\)\}
```

PYMOL API

```
cmd.save\(string file, string selection, int state, string format\)
```

NOTES

```
The file format is automatically chosen if the extesion is one of
the supported output formats: pdb, pqr, mol, sdf, pkl, pkla, mmd, out,
dat, mmod, cif, pov, png, pse, psw, aln, fasta, obj, mtl, wrl, dae, idtf,
or mol2.

If the file format is not recognized, then a PDB file is written
by default.

For molecular files and where applicable and supported:

\* if state = -1 \(default\), then only the current state is written.

\* if state = 0, then a multi-state output file is written.
```

SEE ALSO

```
[load](https://pymol.org/pymol-command-ref.html#load), get\_model
```

api: pymol.exporting.save

---

scene

DESCRIPTION

```
"scene" saves and restores scenes.  A scene consists of the camera
view, all object activity information, all atom-wise visibilities,
all atom-wise colors, all representations, the global frame index,
and may contain a text message to display on playback.
```

USAGE

```
scene \[key \[,action \[, message, \[ new\_key=new-key-value \]\]\]\]
```

ARGUMENTS

```
key = string, new, auto, or \*: use new for an automatically
numbered new scene, use auto for the current scene \(if one
exists\), and use \* for all scenes \(clear and recall actions only\).

action = store, recall, insert\_after, insert\_before, next,
previous, update, rename, or clear: \(default = recall\).  If
rename, then a new\_key argument must be explicitly defined.

message = string: a text message to display with the scene.

new\_key = string: the new name for the scene
```

EXAMPLES

```
scene \*

scene F1, store
scene F2, store, Please note the critical hydrogen bond shown in yellow.

scene F1
scene F2

scene F1, rename, new\_key=F5
```

NOTES

```
Scenes F1 through F12 are automatically bound to function keys
provided that "set\_key" has not been used to redefine the behaviour
of the respective key.
```

SEE ALSO

```
[view](https://pymol.org/pymol-command-ref.html#view), [set\_view]
(https://pymol.org/pymol-command-ref.html#set_view), [get\_view]
(https://pymol.org/pymol-command-ref.html#get_view)
```

api: pymol.viewing.scene

---

scene_order

DESCRIPTION

```
"scene\_order" changes the ordering of scenes.
```

USAGE

```
scene\_order names, sort, location
```

ARGUMENTS

```
names = string: a space-separated list of names

sort = yes or no \{default: no\}

location = top, current, or bottom \{default: current\}
```

EXAMPLES

```
scene\_order \*,yes
scene\_order F6 F4 F3
scene\_order 003 006 004, location=top
```

PYMOL API

```
cmd.scene\_order\(string names, string sort, string location\)
```

SEE ALSO

```
[scene](https://pymol.org/pymol-command-ref.html#scene)
```

api: pymol.viewing.scene_order

---

sculpt_activate

DESCRIPTION

```
"sculpt\_activate" enables sculpting for the given object. The current
geometry \(bond lengths, angles, etc.\) of the given state is remembered as
the reference geometry.
```

ARGUMENTS

```
object = str: name of a single object or "all"

state = int: object state or 0 for current state \{default: 0\}
```

SEE ALSO

```
[sculpt\_iterate](https://pymol.org/pymol-command-ref.html#sculpt_iterate),
[sculpt\_deactivate](https://pymol.org/pymol-command-ref.html#sculpt_deactivate)
```

api: pymol.editing.sculpt_activate

---

sculpt_deactivate

DESCRIPTION

```
"sculpt\_deactivate" deactivates sculpting for the given object and
clears the stored restraints.
```

ARGUMENTS

```
object = str: name of a single object or "all"
```

SEE ALSO

```
[sculpt\_activate](https://pymol.org/pymol-command-ref.html#sculpt_activate)
```

api: pymol.editing.sculpt_deactivate

---

sculpt_iterate

DESCRIPTION

```
"sculpt\_iterate" performs a simple energy minimization of atomic
coordinates based on the geometry restraints which were defined with
the "sculpt\_activate" invocation and which are selected in the
"sculpt\_field\_mask" setting. Sculpting currently supports local
geometry restraints and vdw repulsion, but no solvation or
electrostatic effects.
```

ARGUMENTS

```
object = str: name of a single object or "all"

state = int: object state or 0 for current state \{default: 0\}

cycles = int: number of iterations \{default: 10\}
```

SEE ALSO

```
commands: [sculpt\_activate](https://pymol.org/pymol-command-
ref.html#sculpt_activate), [sculpt\_deactivate](https://pymol.org/pymol-command-
ref.html#sculpt_deactivate)
settings: "sculpting" setting, all "sculpt\_\*" settings
```

api: pymol.editing.sculpt_iterate

---

sculpt_purge

DESCRIPTION

```
"sculpt\_purge" is an unsupported feature.
```

api: pymol.editing.sculpt_purge

---

select

DESCRIPTION

```
"select" creates a named atom selection from a
selection-expression.
```

USAGE

```
select name, selection \[, enable \[, quiet \[, merge \[, state \[, domain
\]\]\]\]\]
```

ARGUMENTS

```
name = a unique name for the selection

selection = a selection-expression
```

NOTES

```
If a selection-expression with explicit surrounding parethenses is
provided as the first argument, then the default selection name
is used as the name argument.
```

EXAMPLES

```
select chA, chain A
select \( resn HIS \)
select near142, resi 142 around 5
```

PYMOL API

```
cmd.select\(string name, string selection\)
```

SEE ALSO

```
[delete](https://pymol.org/pymol-command-ref.html#delete)
```

api: pymol.selecting.select

---

set

DESCRIPTION

```
"set" changes global, object, object-state, or per-atom settings.
```

USAGE

```
set name \[,value \[,selection \[,state \]\]\]
```

ARGUMENTS

```
name = string: setting name

value = string: a setting value \{default: 1\}

selection = string: name-pattern or selection-expression
\{default:'' \(global\)\}

state = a state number \{default: 0 \(per-object setting\)\}
```

EXAMPLES

```
set orthoscopic

set line\_width, 3

set surface\_color, white, 1hpv

set sphere\_scale, 0.5, elem C
```

NOTES

```
The default behavior \(with a blank selection\) is global.  If the
selection is "all", then the setting entry in each individual
object will be changed.  Likewise, for a given object, if state is
zero, then the object setting will be modified.  Otherwise, the
setting for the indicated state within the object will be
modified.

If a selection is provided as opposed to an object name, then the
atomic setting entries are modified.

The following per-atom settings are currently implemented.  Others
may seem to be recognized but will have no effect when set on a
per-atom basis.

\* sphere\_color
\* surface\_color
\* mesh\_color
\* label\_color
\* dot\_color
\* cartoon\_color
\* ribbon\_color
\* transparency \(for surfaces\)
\* sphere\_transparency

Note that if you attempt to use the "set" command with a per-bond
setting over a selection of atoms, the setting change will appear
to take, but no change will be observed.  Please use the
"set\_bond" command for per-bond settings.
```

PYMOL API

```
cmd.set\(string name, string value, string selection, int state,
        int updates, int quiet\)
```

SEE ALSO

```
[get](https://pymol.org/pymol-command-ref.html#get), [set\_bond]
(https://pymol.org/pymol-command-ref.html#set_bond)
```

api: pymol.setting.set

---

set_atom_property

DESCRIPTION

```
Set an atom-level property
```

USAGE

```
set\_atom\_property name, value \[, selection \[, state \[, proptype \]\]\]
```

ARGUMENTS

```
name = string: Name of the property

value = str/int/float/bool: Value to be set

selection = string: a selection-expression
\{default: all\}

proptype = int: The type of the property, -1=auto, 1=bool, 2=int,
3=float, 5=color, 6=str. Type -1 will detect int \(digits only\), float,
and bool \(true/false/yes/no\). \{default: -1\}

state = int: Object state, 0 for all states, -1 for current state
\{default: 0\}
```

EXAMPLE

```
set\_atom\_property myfloatprop,  1.23, elem C
set\_atom\_property myfloatprop,  1234, elem N, proptype=3
set\_atom\_property myboolprop,   TRUE, elem O
set\_atom\_property mystrprop,   false, elem O, proptype=6
set\_atom\_property mystrprop, One Two, elem C
iterate all, print\(elem, p.all\)
alter all, p.myboolprop = True
alter all, p.myfloatprop = None # clear
```

SEE ALSO

```
[set\_property](https://pymol.org/pymol-command-ref.html#set_property), [iterate]
(https://pymol.org/pymol-command-ref.html#iterate), [alter]
(https://pymol.org/pymol-command-ref.html#alter)
```

api: pymol.properties.set_atom_property

---

set_bond

DESCRIPTION

```
"set\_bond" changes per-bond settings for all bonds which exist
between two selections of atoms.
```

USAGE

```
set\_bond name, value, selection1 \[, selection2 \]
```

ARGUMENTS

```
name = string: name of the setting

value = string: new value to use

selection1 = string: first set of atoms

selection2 = string: seconds set of atoms \{default: \(selection1\)\}
```

EXAMPLE

```
set\_bond stick\_transparency, 0.7, \*/n+c+ca+o
```

NOTES

```
The following per-bond settings are currently implemented.  Others
may seem to be recognized but will currently have no effect when
set at the per-bond level.

\* valence
\* line\_width
\* line\_color
\* stick\_radius
\* stick\_color
\* stick\_transparency

Note that if you attempt to use the "set" command with a per-bond
setting over a selection of atoms, the setting change will appear
to take, but no change will be observed.
```

PYMOL API

```
cmd.set\_bond \( string name, string value,
               string selection1,
               string selection2,
               int state, int updates, log=0, quiet=1\)
```

api: pymol.setting.set_bond

---

set_color

DESCRIPTION

```
"set\_color" defines a new color using the red, green, and blue
\(RGB\) color components.
```

USAGE

```
set\_color name, rgb
```

ARGUMENTS

```
name = string: name for the new or existing color

rgb = list of numbers: \[red, green, blue\] each and all in the range
\(0.0, 1.0\) or \(0, 255\)
```

EXAMPLES

```
set\_color red, \[ 1.0, 0.0, 0.0 \]

set\_color yellow, \[ 255, 255, 0 \]
```

NOTES

```
PyMOL automatically infers the range based on the input arguments.

It may be necessary to issue "recolor" command in order to force
recoloring of existing objects.
```

SEE ALSO

```
[recolor](https://pymol.org/pymol-command-ref.html#recolor)
```

PYMOL API

```
cmd.set\_color\(string name, list-of-numbers rgb, int mode \)
```

api: pymol.viewing.set_color

---

set_dihedral

DESCRIPTION

```
"set\_dihedral" changes the dihedral angle formed between the four
bonded atoms provided.  The atoms must be acyclic.
```

USAGE

```
set\_dihedral atom1, atom2, atom3, atom4, angle \[, state \[, quiet \]\]
```

NOTES

```
Because set\_dihedral uses the molecular editing capability,
numbered "pk" atom selections \(if any\) will be redefined by this
operation.
```

PYMOL API

```
cmd.set\_dihedral\(string atom1, string atom2, string atom3, string atom4,
                  float angle, int state, int quiet\)
```

api: pymol.editing.set_dihedral

---

set_geometry

DESCRIPTION

```
"set\_geometry" changes PyMOL's assumptions about the proper valence
and geometry of atoms in the selection.
```

USAGE

```
set\_geometry selection, geometry, valence
```

NOTES

```
Immature functionality. See code for details.
```

PYMOL API

```
cmd.set\_geometry\(string selection, int geometry, int valence\)
```

SEE ALSO

```
[remove](https://pymol.org/pymol-command-ref.html#remove), [attach]
(https://pymol.org/pymol-command-ref.html#attach), [fuse]
(https://pymol.org/pymol-command-ref.html#fuse), [bond](https://pymol.org/pymol-
command-ref.html#bond), [unbond](https://pymol.org/pymol-command-
ref.html#unbond)
```

api: pymol.editing.set_geometry

---

set_key

DESCRIPTION

```
"set\_key" binds a specific python function to a key press.

New in PyMOL 1.6.1: second argument can also be a string in PyMOL
command syntax.
```

USAGE

```
set\_key key, command
```

EXAMPLE

```
set\_key F1, as cartoon, polymer; as sticks, organic
```

PYMOL API (ONLY)

```
cmd.set\_key\( string key, function fn, tuple arg=\(\), dict kw=\{\}\)
```

PYTHON EXAMPLE

```
from pymol import cmd

def color\_blue\(object\): cmd.color\("blue",object\)

cmd.set\_key\( 'F1' , color\_blue, \( "object1" \) \)
// would turn object1 blue when the F1 key is pressed and

cmd.set\_key\( 'F2' , color\_blue, \( "object2" \) \)
// would turn object2 blue when the F2 key is pressed.

cmd.set\_key\( 'CTRL-C' , cmd.zoom \)
cmd.set\_key\( 'ALT-A' , cmd.turn, \('x',90\) \)
```

KEYS WHICH CAN BE REDEFINED

```
F1 to F12
left, right, pgup, pgdn, home, insert
CTRL-A to CTRL-Z
ALT-0 to ALT-9, ALT-A to ALT-Z
```

SEE ALSO

```
[button](https://pymol.org/pymol-command-ref.html#button), [alias]
(https://pymol.org/pymol-command-ref.html#alias)
```

api: pymol.controlling.set_key

---

set_name

DESCRIPTION

```
"set\_name" changes the name of an object or selection.
```

USAGE

```
set\_name old\_name, new\_name
```

PYMOL API

```
cmd.set\_name\(string old\_name, string new\_name\)
```

api: pymol.editing.set_name

---

set_property

DESCRIPTION

```
Set an object-level property
```

USAGE

```
set\_property name, value \[, object \[, state \[, proptype \]\]\]
```

ARGUMENTS

```
name = string: Name of the property

value = str/int/float/bool: Value to be set

object = string: Space separated list of objects or \* for all objects
\{default: \*\}

proptype = int: The type of the property, -1=auto, 1=bool, 2=int,
3=float, 5=color, 6=str. Type -1 will detect int \(digits only\), float,
and bool \(true/false/yes/no\). \{default: -1\}

state = int: Object state, 0 for all states, -1 for current state
\{default: 0\}
```

EXAMPLE

```
fragment ala
set\_property myfloatprop, 1234, ala, proptype=3
get\_property myfloatprop, ala
```

SEE ALSO

```
[get\_property](https://pymol.org/pymol-command-ref.html#get_property),
[get\_property\_list](https://pymol.org/pymol-command-
ref.html#get_property_list), [set\_atom\_property](https://pymol.org/pymol-
command-ref.html#set_atom_property)
```

api: pymol.properties.set_property

---

set_symmetry

DESCRIPTION

```
"set\_symmetry" defines or redefines the crystal and spacegroup
parameters for a molecule or map object.
```

USAGE

```
set\_symmetry selection, a, b, c, alpha, beta, gamma, spacegroup
```

ARGUMENTS

```
selection = str: object name pattern
```

PYMOL API

```
cmd.set\_symmetry\(string selection, float a, float b, float c,
      float alpha, float beta, float gamma, string spacegroup\)
```

api: pymol.editing.set_symmetry

---

set_title

DESCRIPTION

```
"set\_title" attaches a text string to the state of a particular
object which can be displayed next to the object name when that
state is active.  This is useful for display the energies of a set
of conformers.
```

USAGE

```
set\_title object, state, text
```

PYMOL API

```
cmd.set\_title\(string object, int state, string text\)
```

api: pymol.editing.set_title

---

set_view

DESCRIPTION

```
"set\_view" sets viewing information for the current scene,
including the rotation matrix, position, origin of rotation,
clipping planes, and the orthoscopic flag.
```

USAGE

```
set\_view \[ view \]
```

EXAMPLE

```
set\_view \(\\
    0.999876618,   -0.000452542,   -0.015699286,\\
    0.000446742,    0.999999821,   -0.000372844,\\
    0.015699454,    0.000365782,    0.999876678,\\
    0.000000000,    0.000000000, -150.258514404,\\
    11.842411041,   20.648729324,    8.775371552,\\
    118.464958191,  182.052062988,    0.000000000 \)
```

PYMOL API

```
cmd.set\_view\(string-or-sequence view\)
```

SEE ALSO

```
[get\_view](https://pymol.org/pymol-command-ref.html#get_view)
```

api: pymol.viewing.set_view

---

show

DESCRIPTION

```
"show" turns on representations for objects and selections.
```

USAGE

```
show \[ representation \[, selection \]\]
```

ARGUMENTS

```
representation = lines, spheres, mesh, ribbon, cartoon, sticks,
   dots, surface, labels, extent, nonbonded, nb\_spheres, slice,
   extent, slice, dashes, angles, dihedrals, cgo, cell, callback,
   or everything

selection = string: a selection-expression or name-pattern
```

NOTES

```
With no arguments, "show" alone turns on lines for all bonds and
nonbonded for all atoms in all molecular objects.
```

EXAMPLES

```
show
show ribbon
show lines, \(name CA+C+N\)
```

SEE ALSO

```
[hide](https://pymol.org/pymol-command-ref.html#hide), [enable]
(https://pymol.org/pymol-command-ref.html#enable), [disable]
(https://pymol.org/pymol-command-ref.html#disable)
```

api: pymol.viewing.show

---

skip

DESCRIPTION

```
"skip" delimits a block of commands that are skipped instead of
being executed.
```

EXAMPLE

```
skip

# the following command will not be executed
color blue, all

skip end
```

NOTES

```
If the "skip" command is commented out, the subsequent "skip end"
can be left in place, and will have no effect upon execution of
subsequent commands.
```

SEE ALSO

```
[abort](https://pymol.org/pymol-command-ref.html#abort), [embed]
(https://pymol.org/pymol-command-ref.html#embed), [python]
(https://pymol.org/pymol-command-ref.html#python)
```

api: pymol.helping.skip

---

slice_new

DESCRIPTION

```
"slice\_map" creates a slice object from a map object.
```

USAGE

```
slice\_map name, map, \[opacity, \[resolution, \[state, \[source\_state\]\]\]\]
```

ARGUMENTS

```
name = the name for the new slice object.

map = the name of the map object to use for computing the slice.

opacity = opacity of the new slice \(default=1\)

resolution = the number of pixels per sampling \(default=5\)

state = the state into which the object should be loaded \(default=1\)
    \(set state=0 to append new mesh as a new state\)

source\_state = the state of the map from which the object should be loaded \
(default=0\)
```

SEE ALSO

```
[isomesh](https://pymol.org/pymol-command-ref.html#isomesh), [isodot]
(https://pymol.org/pymol-command-ref.html#isodot), [load]
(https://pymol.org/pymol-command-ref.html#load)
```

api: pymol.creating.slice_new

---

smooth

DESCRIPTION

```
"smooth" performs a window average of coordinate states.
```

USAGE

```
smooth \[ selection \[, passes \[, window \[, first \[, last \[,
ends\]\]\]\]\]\]
```

ARGUMENTS

```
ends = 0 or 1: controls whether or not the end states are also smoothed
using a weighted asymmetric window
```

NOTES

```
This type of averaging is often used to suppress high-frequency
vibrations in a molecular dynamics trajectory.

This function is not memory efficient.  For reasons of
flexibility, it uses two additional copies of every atomic
coordinate for the calculation.  If you are memory-constrained in
visualizing MD trajectories, then you may want to use an external
tool such as ptraj to perform smoothing before loading coordinates
into PyMOL.
```

SEE ALSO

```
[load\_traj](https://pymol.org/pymol-command-ref.html#load_traj)
```

api: pymol.editing.smooth

---

sort

DESCRIPTION

```
"sort" reorders atoms in the structure.  It usually only necessary
to run this routine after an "alter" command which has modified the
names of atom properties.  Without an argument, sort will resort
all atoms in all objects.
```

USAGE

```
sort \[object\]
```

PYMOL API

```
cmd.sort\(string object\)
```

SEE ALSO

```
[alter](https://pymol.org/pymol-command-ref.html#alter)
```

api: pymol.editing.sort

---

space

DESCRIPTION

```
"space" selects a color palette \(or color space\).
```

USAGE

```
space space \[, gamma\]
```

ARGUMENTS

```
space = rgb, cmyk, or pymol: \{default: rgb\}

gamma = floating point gamma transformation
```

EXAMPLES

```
space rgb
space cmyk
space pymol
```

NOTES

```
Whereas computer displays use the RGB color space, computer
printers typically use the CMYK color space.  The two spaces are
non-equivalent, meaning that certain RGB colors cannot be
expressed in the CMYK space and vice-versa.  And as a result,
molecular graphics images prepared using RGB often turn out poorly
when converted to CMYK, with purplish blues or yellowish greens.

"space cmyk" forces PyMOL to restrict its use of the RGB color
space to subset that can be reliably converted to CMYK using
common tools such as Adobe Photoshop.  Thus, what you see on the
screen is much closer to what you will get in print.

Analog video systems as well as digital video compression codecs
based on the YUV color space also have incompatibilities with RGB.
Oversaturated colors usually cause the most problems.

Although PyMOL lacks "space yuv", "space pymol" will help PyMOL
avoid oversaturated colors can cause problems when exporting
animations to video.
```

PYMOL API

```
cmd.space\(string space, float gamma\)
```

SEE ALSO

```
[color](https://pymol.org/pymol-command-ref.html#color)
```

api: pymol.importing.space

---

spawn

DESCRIPTION

```
"spawn" launches a Python script in a new thread which will run
concurrently with the PyMOL interpreter. It can be run in its own
namespace \(like a Python module, default\), a local name space, or
in the global namespace.
```

USAGE

```
spawn file \[, namespace \]
```

NOTES

```
The default namespace for spawn is "module".

The best way to spawn processes at startup is to use the -l option
\(see "help launching"\).
```

SEE ALSO

```
[run](https://pymol.org/pymol-command-ref.html#run)
```

api: pymol.parsing.spawn

---

spectrum

DESCRIPTION

```
"spectrum" colors atoms with a spectrum of colors based on an atomic
property.
```

USAGE

```
spectrum \[expression \[, palette \[, selection \[, minimum \[, maximum \[, byres
\]\]\]\]\]\]
```

ARGUMENTS

```
expression = count, b, q, pc or properties\["key"\]: respectively, atom
count, temperature factor, occupancy, partial charge or a user defined
numeric atom property \{default: count\}

palette = string: palette name or space separated list of colors
\{default: rainbow\}

selection = string: atoms to color \{default: \(all\)\}

minimum = float: \{default: None \(automatic\)\}

maximum = float: \{default: None \(automatic\)\}

byres = integer: controls whether coloring is applied per-residue \{default: 0\}
```

EXAMPLES

```
spectrum b, blue\_red, minimum=10, maximum=50

spectrum count, rainbow\_rev, chain A, byres=1

spectrum properties\["some\_score"\], red yellow white
```

NOTES

```
Available palettes include:

   blue\_green blue\_magenta blue\_red blue\_white\_green
   blue\_white\_magenta blue\_white\_red blue\_white\_yellow blue\_yellow
   cbmr cyan\_magenta cyan\_red cyan\_white\_magenta cyan\_white\_red
   cyan\_white\_yellow cyan\_yellow gcbmry green\_blue green\_magenta
   green\_red green\_white\_blue green\_white\_magenta green\_white\_red
   green\_white\_yellow green\_yellow green\_yellow\_red magenta\_blue
```

```
magenta\_cyan magenta\_green magenta\_white\_blue
magenta\_white\_cyan magenta\_white\_green magenta\_white\_yellow
magenta\_yellow rainbow rainbow2 rainbow2\_rev rainbow\_cycle
rainbow\_cycle\_rev rainbow\_rev red\_blue red\_cyan red\_green
red\_white\_blue red\_white\_cyan red\_white\_green red\_white\_yellow
red\_yellow red\_yellow\_green rmbc yellow\_blue yellow\_cyan
yellow\_cyan\_white yellow\_green yellow\_magenta yellow\_red
yellow\_white\_blue yellow\_white\_green yellow\_white\_magenta
yellow\_white\_red yrmbcg
```

PYMOL API

```
def spectrum\(string expression, string palette,
             string selection, float minimum, float maximum,
             int byres, int quiet\)
```

api: pymol.viewing.spectrum

---

spheroid

DESCRIPTION

```
"spheroid" averages trajectory frames together to create
an ellipsoid-like approximation of the actual anisotropic
motion exhibited by the atom over a series of trajectory frames.
```

USAGE

```
spheroid object,average

average = number of states to average for each resulting spheroid state
```

api: pymol.experimenting.spheroid

---

splash

UNDOCUMENTED

api: pymol.commanding.splash

---

split_chains

DESCRIPTION

```
Create a single object for each chain in selection
```

SEE ALSO

```
[split\_states](https://pymol.org/pymol-command-ref.html#split_states)
```

api: pymol.editing.split_chains

---

split_states

DESCRIPTION

```
"split\_states" separates a multi-state molecular object into a set
of single-state molecular objects.
```

USAGE

```
split\_states object \[, first \[, last \[, prefix \]\]\]
```

EXAMPLE

```
load docking\_hits.sdf
split\_states docking\_hits, prefix=hit
delete docking\_hits
```

SEE ALSO

```
[join\_states](https://pymol.org/pymol-command-ref.html#join_states)
```

api: pymol.editing.split_states

---

stereo

DESCRIPTION

```
"stereo" activates or deactives stereo mode.
```

USAGE

```
stereo \[toggle\]
```

ARGUMENTS

```
toggle = on, off, crosseye, walleye, quadbuffer, sidebyside, geowall, or openvr
```

EXAMPLES

```
stereo on
stereo off
stereo crosseye
```

NOTES

```
"quadbuffer" is the default stereo mode if hardware stereo is available.
otherwise, "crosseye" is the default.
```

PYMOL API

```
cmd.stereo\(string toggle\)
```

api: pymol.viewing.stereo

---

super

DESCRIPTION

```
    "super" performs a residue-based pairwise alignment followed by a
    structural superposition, and then carries out zero or more cycles
    of refinement in order to reject outliers.
```

USAGE

```
    super mobile, target \[, object=name \]
```

NOTES

```
    By adjusting various parameters, the nature of the initial
    alignment can be modified to include or exclude various factors
    including sequence similarity, main chain path, secondary \&
    tertiary structure, and current coordinates.
```

EXAMPLE

```
    super protA////CA, protB////CA, object=supeAB
```

SEE ALSO

```
    [align](https://pymol.org/pymol-command-ref.html#align), [pair\_fit]
(https://pymol.org/pymol-command-ref.html#pair_fit), [fit]
(https://pymol.org/pymol-command-ref.html#fit), [rms](https://pymol.org/pymol-
command-ref.html#rms), [rms\_cur](https://pymol.org/pymol-command-
ref.html#rms_cur), [intra\_rms](https://pymol.org/pymol-command-
ref.html#intra_rms), [intra\_rms\_cur](https://pymol.org/pymol-command-
ref.html#intra_rms_cur)
```

api: pymol.fitting.super

---

symexp

DESCRIPTION

```
"symexp" creates all symmetry-related objects for the specified
object that occur within a cutoff about an atom selection.
```

USAGE

```
symexp prefix, object, selection, cutoff
```

NOTES

```
The newly objects are labeled using the prefix provided along with
their crystallographic symmetry operation and translation.
```

SEE ALSO

```
[load](https://pymol.org/pymol-command-ref.html#load)
```

api: pymol.creating.symexp

---

symmetry_copy

DESCRIPTION

```
"symmetry\_copy" copies symmetry information from one object to another.
```

USAGE

```
symmetry\_copy source\_name, target\_name, source\_state, target\_state
```

ARGUMENTS

```
source\_name = str: object name
target\_name = str: object name pattern
source\_state = int: object state \(maps only\)
target\_state = int: object state \(maps only\)
```

NOTES

```
Molecular objects don't support individual states yet.
```

api: pymol.editing.symmetry_copy

---

system

DESCRIPTION

```
"system" executes a command in a subshell under Unix or Windows.
```

USAGE

```
system command
```

PYMOL API

```
cmd.system\(string command,int async=0\)
```

NOTES

```
async can only be specified from the Python level \(not the command language\)

if async is 0 \(default\), then the result code from "system" is returned in r

if async is 1, then the command is run in a separate thread whose object is
returned
```

SEE ALSO

```
[ls](https://pymol.org/pymol-command-ref.html#ls), [cd](https://pymol.org/pymol-
command-ref.html#cd), [pwd](https://pymol.org/pymol-command-ref.html#pwd)
```

api: pymol.externing.system

---

toggle

DESCRIPTION

```
"toggle" toggles the visibility of a representation within a
selection.
```

USAGE

```
toggle \[ representation \[, selection \]\]
```

ARGUMENTS

```
representation = string: named representation \{default: lines\}

selection = string: atom selection \{default: all\}
```

NOTES

```
If the representation is enabled for any atom in the selection, it will
be turned off.
```

PYMOL API

```
cmd.toggle\(string representation, string selection\)
```

SEE ALSO

```
[show](https://pymol.org/pymol-command-ref.html#show), [hide]
(https://pymol.org/pymol-command-ref.html#hide)
```

api: pymol.viewing.toggle

torsion

DESCRIPTION

```
"torsion" rotates the torsion on the bond currently
picked for editing.  The rotated fragment will correspond
to the first atom specified when picking the bond \(or the
nearest atom, if picked using the mouse\).
```

USAGE

```
torsion angle
```

PYMOL API

```
cmd.torsion\( float angle \)
```

SEE ALSO

```
[edit](https://pymol.org/pymol-command-ref.html#edit), [unpick]
(https://pymol.org/pymol-command-ref.html#unpick), [remove\_picked]
(https://pymol.org/pymol-command-ref.html#remove_picked), [cycle\_valence]
(https://pymol.org/pymol-command-ref.html#cycle_valence)
```

api: pymol.editing.torsion

---

translate

DESCRIPTION

```
"translate" translates the atomic coordinates of atoms in a
selection.  Alternatively, is modifies the matrix associated with
a particular object or object-state.
```

USAGE

```
translate vector \[, selection \[, state \[, camera \[, object \]\]\]\]
```

ARGUMENTS

```
vector = float vector: translation vector

selection = string: atoms whose coordinates should be modified \{default: all\}

state > 0: only the indicated state is modified

state = 0: all states are modified

state = -1: only current state is modified \{default\}

camera = 0 or 1: is the vector in camera coordinates\? \{default: 1 \(yes\)\}

object = string: object name \(only if rotating object matrix\) \{default:
None\}
```

PYMOL API

```
cmd.translate\(list vector, string selection, int state, int
              camera, string object\)
```

EXAMPLES

```
translate \[1,0,0\], name CA
```

NOTES

```
"translate" can be used to translate the atomic coordinates of a
molecular object.  Behavior differs depending on whether or not
the "object" parameter is specified.

If object is None, then translate translates atomic coordinates
according to the vector provided for the selection and in the state
provided.  All representation geometries will need to be
regenerated to reflect the new atomic coordinates.

If object is set to an object name, then selection is ignored and
instead of translating the atomic coordinates, the object's
overall representation display matrix is modified.  This option is
for use in animations only.

The "camera" option controls whether the camera or the model's
axes are used to interpret the translation vector.
```

api: pymol.editing.translate

---

turn

DESCRIPTION

```
"turn" rotates the camera about one of the three primary axes,
centered at the origin.
```

USAGE

```
turn axis, angle
```

EXAMPLES

```
turn x, 90
turn y, 45
```

PYMOL API

```
cmd.turn\(string axis, float angle\)
```

SEE ALSO

```
[move](https://pymol.org/pymol-command-ref.html#move), [rotate]
(https://pymol.org/pymol-command-ref.html#rotate), [translate]
(https://pymol.org/pymol-command-ref.html#translate), [zoom]
(https://pymol.org/pymol-command-ref.html#zoom), [center]
(https://pymol.org/pymol-command-ref.html#center), [clip]
(https://pymol.org/pymol-command-ref.html#clip)
```

api: pymol.viewing.turn

---

unbond

DESCRIPTION

```
"unbond" removes all bonds between two selections.
```

USAGE

```
unbond atom1,atom2
```

ARGUMENTS

```
atom1 = string \{default: \(pk1\)\}

atom2 = string \{default: \(pk2\)\}
```

PYMOL API

```
cmd.unbond\(selection atom1, selection atom2\)
```

SEE ALSO

```
[bond](https://pymol.org/pymol-command-ref.html#bond), [fuse]
(https://pymol.org/pymol-command-ref.html#fuse), [remove\_picked]
(https://pymol.org/pymol-command-ref.html#remove_picked), [attach]
(https://pymol.org/pymol-command-ref.html#attach), detach, [replace]
(https://pymol.org/pymol-command-ref.html#replace)
```

api: pymol.editing.unbond

---

undo

DESCRIPTION

```
"undo" restores the previous conformation of the object currently
being edited.
```

USAGE

```
undo
```

SEE ALSO

```
[redo](https://pymol.org/pymol-command-ref.html#redo), push\_undo
```

api: pymol.editing.undo

---

ungroup

DESCRIPTION

```
"ungroup" removes an object from a group object, returning it to
the top level.
```

USAGE

```
ungroup name
```

SEE ALSO

```
[group](https://pymol.org/pymol-command-ref.html#group)
```

api: pymol.creating.ungroup

---

uniquify

DESCRIPTION

```
Make \`identifier\` unique with respect to reference selection.
```

ARGUMENTS

```
identifier = str: atom identifier \(chain, segi, etc.\)

selection = str: atom selection to modify

reference = str: atom selection whose identifiers must not be
present in the first selection \{default: \!selection\}
```

EXAMPLE

```
fetch 1a00 1hbb, async=0
uniquify chain, 1hbb
# 1hbb now has chains E,F,G,H
```

api: pymol.editing.uniquify

---

unmask

DESCRIPTION

```
"unmask" reverses the effect of "mask" on the indicated atoms.
```

PYMOL API

```
cmd.unmask\( string selection="\(all\)" \)
```

USAGE

```
unmask \(selection\)
```

SEE ALSO

```
[mask](https://pymol.org/pymol-command-ref.html#mask), [protect]
(https://pymol.org/pymol-command-ref.html#protect), [deprotect]
(https://pymol.org/pymol-command-ref.html#deprotect), [mouse]
(https://pymol.org/pymol-command-ref.html#mouse)
```

api: pymol.controlling.unmask

---

unpick

DESCRIPTION

```
"unpick" deletes the special "pk" atom selections \(pk1, pk2, etc.\)
used in atom picking and molecular editing.
```

USAGE

```
unpick
```

PYMOL API

```
cmd.unpick\(\)
```

SEE ALSO

```
[edit](https://pymol.org/pymol-command-ref.html#edit)
```

api: pymol.editing.unpick

---

unset

DESCRIPTION

```
"unset" clear non-global settings and zeros out global settings.

WARNING: The behavior for global settings is subject to change.
To set a setting to zero, do "set settingname, 0".
```

USAGE

```
unset name \[,selection \[,state \]\]
```

EXAMPLE

```
unset orthoscopic

unset surface\_color, 1hpv

unset sphere\_scale, elem C
```

NOTES

```
If selection is not provided, unset changes the named global
setting to a zero or off value.

If a selection is provided, then "unset" undefines per-object,
per-state, or per-atom settings.
```

PYMOL API

```
cmd.unset\(string name, string selection, int state, int updates,
           int log\)
```

SEE ALSO

```
[set](https://pymol.org/pymol-command-ref.html#set), [set\_bond]
(https://pymol.org/pymol-command-ref.html#set_bond)
```

api: pymol.setting.unset

---

unset_bond

DESCRIPTION

```
"unset\_bond" removes a per-bond setting for a given set of bonds.
```

USAGE

```
unset name \[,selection \[, selection \[,state \]\]\]
```

api: pymol.setting.unset_bond

---

unset_deep

DESCRIPTION

```
Unset all object, object-state, atom, and bond level settings.

Note: Does currently NOT unset atom-state level settings. Check for
atom-state level settings with:
PyMOL> iterate\_state 1, \*, print\(list\(s\)\)
Unset e.g. atom-state level "label\_screen\_point" \(index 728\) with:
PyMOL> alter\_state 1, \*, del s\[728\]
```

ARGUMENTS

```
settings = str: space separated list of setting names or empty string
for all settings \{default: \}

object = str: name of one object or \* for all objects \{default: \*\}
```

api: pymol.setting.unset_deep

---

update

DESCRIPTION

```
"update" transfers coordinates from one selection to another.
```

USAGE

```
update \(target-selection\),\(source-selection\)
```

EXAMPLES

```
update target,\(variant\)
```

NOTES

```
Currently, this applies across all pairs of states.  Fine
control will be added later.
```

SEE ALSO

```
[load](https://pymol.org/pymol-command-ref.html#load)
```

api: pymol.editing.update

---

util.cbab

Wrapper around "color atomic"

api: pymol.util.cbab

---

util.cbac

Wrapper around "color atomic"

api: pymol.util.cbac

---

util.cbag

Wrapper around "color atomic"

api: pymol.util.cbag

---

util.cbak

Wrapper around "color atomic"

api: pymol.util.cbak

---

util.cbam

Wrapper around "color atomic"

api: pymol.util.cbam

---

util.cbao

Wrapper around "color atomic"

api: pymol.util.cbao

---

util.cbap

Wrapper around "color atomic"

api: pymol.util.cbap

util.cbas

Wrapper around "color atomic"

api: pymol.util.cbas

util.cbaw

Wrapper around "color atomic"

api: pymol.util.cbaw

util.cbay

Wrapper around "color atomic"

api: pymol.util.cbay

util.cbc

Color all chains a different color

api: pymol.util.cbc

util.chainbow

Color all chains in rainbow

api: pymol.util.chainbow

util.cnc

Wrapper around "color atomic"

api: pymol.util.cnc

util.rainbow

Legacy spectrum coloring routine. Don't use.

```
Use instead: spectrum
```

api: pymol.util.rainbow

util.ss

Legacy secondary structure assignment routine. Don't use.

```
Use instead: dss
```

api: pymol.util.ss

---

valence

DESCRIPTION

```
"valence" modifies the valences of all existing bonds formed
between two atom selections.
```

USAGE

```
valence 2, \(name C\), \(name O\)
```

PYMOL API

```
cmd.valence\(string selection1, selection2\)
```

SEE ALSO

```
[unbond](https://pymol.org/pymol-command-ref.html#unbond), [fuse]
(https://pymol.org/pymol-command-ref.html#fuse), [attach]
(https://pymol.org/pymol-command-ref.html#attach), [replace]
(https://pymol.org/pymol-command-ref.html#replace), [remove\_picked]
(https://pymol.org/pymol-command-ref.html#remove_picked)
```

api: pymol.editing.valence

---

vdw_fit

DESCRIPTION

```
"vdw\_fit" is an unsupported feature.
```

api: pymol.editing.vdw_fit

---

view

DESCRIPTION

```
"view" saves and restore camera views.
```

USAGE

```
view key \[, action \[, animate\]\]
```

ARGUMENTS

```
key = string or \*

action = store, recall, clear: \{default: recall\}
```

NOTES

```
Views F1 through F12 are automatically bound to function keys
provided that "set\_key" has not been used to redefine the
behaviour of the respective key, and that a "scene" has not been
defined for that key.
```

EXAMPLES

```
view 0, store
view 0
```

PYMOL API

```
cmd.view\(string key, string action\)
```

SEE ALSO

```
[scene](https://pymol.org/pymol-command-ref.html#scene), [set\_view]
(https://pymol.org/pymol-command-ref.html#set_view), [get\_view]
(https://pymol.org/pymol-command-ref.html#get_view)
```

api: pymol.viewing.view

---

viewport

DESCRIPTION

```
"viewport" changes the size of the graphics display area.
```

USAGE

```
viewport width, height
```

PYMOL API

```
cmd.viewport\(int width, int height\)
```

api: pmg_qt.pymol_qt_gui.viewport

---

volume

DESCRIPTION

```
"volume" creates a volume object from a map object.
```

USAGE

```
volume name, map \[, ramp \[, selection \[, buffer \[, state \[, carve
\]\]\]\]\]
```

ARGUMENTS

```
name = the name for the new volume object.

map = the name of the map object to use for computing the volume.

ramp = str: named color ramp \{default: \}

selection = an atom selection about which to display the mesh with
    an additional "buffer" \(if provided\).

carve = a radius about each atom in the selection for which to
    include density. If "carve" is not provided, then the whole
    brick is displayed.
```

NOTES

```
If the volume object already exists, then the new volume will
overwrite the existing object.
```

EXAMPLE

```
fetch 1oky, async=0
fetch 1oky, type=2fofc, async=0
volume 1okyVol, 1oky\_2fofc
```

SEE ALSO

```
[map\_new](https://pymol.org/pymol-command-ref.html#map_new), [isosurface]
(https://pymol.org/pymol-command-ref.html#isosurface), [isomesh]
(https://pymol.org/pymol-command-ref.html#isomesh), [volume\_color]
(https://pymol.org/pymol-command-ref.html#volume_color), [volume\_ramp\_new]
(https://pymol.org/pymol-command-ref.html#volume_ramp_new)
```

api: pymol.creating.volume

---

volume_color

DESCRIPTION

```
Set or get the volume colors.
```

ARGUMENTS

```
name = str: volume object name

ramp = str, list or empty: named ramp, space delimited string or list
with \(x, color, alpha, ...\) or \(x, r, g, b, alpha, ...\) values. If empty,
get
the current volume colors.
```

EXAMPLE

```
fetch 1a00, map, type=2fofc
volume vol, map
volume\_color vol, .8 cyan 0. 1. blue .3 2. yellow .3
```

api: pymol.colorramping.volume_color

---

volume_panel

DESCRIPTION

```
Open an interactive volume ramp panel
```

ARGUMENTS

```
name = str: name of volume object
```

api: pymol.colorramping.volume_panel

---

volume_ramp_new

DESCRIPTION

```
Register a named volume ramp which can be used as a preset
when creating or coloring volumes. The name will appear in the
internal menu at "A > volume" and "C".
```

ARGUMENTS

```
name = string: name of the new ramp

ramp = list: space delimited list of value, color, alpha
```

EXAMPLE

```
volume\_ramp\_new pink1sigma, \\
   0.9 violet 0.0 \\
   1.0 magenta 0.3 \\
   1.5 pink 0.0
```

SEE ALSO

```
[volume](https://pymol.org/pymol-command-ref.html#volume), [volume\_color]
(https://pymol.org/pymol-command-ref.html#volume_color)
```

api: pymol.colorramping.volume_ramp_new

---

window

DESCRIPTION

```
"window" controls the visibility of PyMOL's output window
```

USAGE

```
window \[ action \[, x \[, y \[, width \[, height \]\]\]\]\]
```

PYMOL API

```
cmd.window\(string action, int x, int y, int width, int height\)
```

api: pymol.viewing.window

---

wizard

DESCRIPTION

```
"wizard" launches on of the built-in wizards.  There are special
Python scripts which work with PyMOL in order to obtain direct user
interaction and easily peform complicated tasks.
```

USAGE

```
wizard name
```

PYMOL API

```
cmd.wizard\(string name\)
```

EXAMPLE

```
wizard distance  # launches the distance measurement wizard
```

api: pymol.wizarding.wizard

---

zoom

DESCRIPTION

```
"zoom" scales and translates the window and the origin to cover the
atom selection.
```

USAGE

```
zoom \[ selection \[, buffer \[, state \[, complete \[, animate \]\]\]\]\]
```

EXAMPLES

```
zoom
zoom complete=1
zoom 142/, animate=3
zoom \(chain A\)
```

ARGUMENTS

```
selection = string: selection-expression or name pattern \{default: all\}

buffer = float: distance  \{default: 0\}

state = 0: uses all coordinate states \{default\}

state = -1: uses only coordinates for the current state

state > 0: uses coordinates for a specific state

complete = 0 or 1: will insure no atoms centers are clipped

animate \< 0: uses the default animation duration

animate = 0: no animation

animate > 0: animates using the provided duration in seconds
```

PYMOL API

```
cmd.zoom\(string selection, float buffer, int state, int complete,
         int animate\)
```

NOTES

```
The zoom command normally tries to guess an optimal zoom level for
visualization, balancing closeness against occasional clipping of
atoms out of the field of view.  You can change this behavior by
setting the complete option to 1, which will guarantee that the
atom positions for the entire selection will fit in the field of
an orthoscopic view.

To absolutely prevent clipping, you may also need to add an
additional buffer \(typically 2 A\) to account for graphical
representations which extend beyond the atom coordinates.
```

SEE ALSO

```
[origin](https://pymol.org/pymol-command-ref.html#origin), [orient]
(https://pymol.org/pymol-command-ref.html#orient), [center]
(https://pymol.org/pymol-command-ref.html#center)
```

api: pymol.viewing.zoom