

Starting Octave

<code>octave</code>	start interactive Octave session
<code>octave file</code>	run Octave on commands in <i>file</i>
<code>octave --help</code>	describe command line options

Stopping Octave

<code>quit</code> or <code>exit</code>	exit Octave
<code>INTERRUPT</code>	(<i>e.g.</i> <code>C-c</code>) terminate current command and return to top-level prompt

Getting Help

<code>help</code>	list all commands and built-in variables
<code>help command</code>	briefly describe <i>command</i>
<code>help -i</code>	use Info to browse Octave manual
<code>help -i command</code>	search for <i>command</i> in Octave manual

Motion in Info

<code>SPC</code> or <code>C-v</code>	scroll forward one screenful
<code>DEL</code> or <code>M-v</code>	scroll backward one screenful
<code>C-l</code>	redraw the display

Node Selection in Info

<code>n</code>	select the next node
<code>p</code>	select the previous node
<code>u</code>	select the ‘up’ node
<code>t</code>	select the ‘top’ node
<code>d</code>	select the directory node
<code><</code>	select the first node in the current file
<code>></code>	select the last node in the current file
<code>g</code>	reads the name of a node and selects it
<code>C-x k</code>	kills the current node

Searching in Info

<code>s</code>	search for a string
<code>C-s</code>	search forward incrementally
<code>C-r</code>	search backward incrementally
<code>i</code>	search index & go to corresponding node
<code>,</code>	go to next match from last ‘i’ command

Command-Line Cursor Motion

<code>C-b</code>	move back one character
<code>C-f</code>	move forward one character
<code>C-a</code>	move the the start of the line
<code>C-e</code>	move to the end of the line
<code>M-f</code>	move forward a word
<code>M-b</code>	move backward a word
<code>C-l</code>	clear screen, reprinting current line at top

Inserting or Changing Text

<code>M-TAB</code>	insert a tab character
<code>DEL</code>	delete character to the left of the cursor
<code>C-d</code>	delete character under the cursor
<code>C-v</code>	add the next character verbatim
<code>C-t</code>	transpose characters at the point
<code>M-t</code>	transpose words at the point

[] surround optional arguments ... show one or more arguments

Killing and Yanking

<code>C-k</code>	kill to the end of the line
<code>C-y</code>	yank the most recently killed text
<code>M-d</code>	kill to the end of the current word
<code>M-DEL</code>	kill the word behind the cursor
<code>M-y</code>	rotate the kill ring and yank the new top

Command Completion and History

<code>TAB</code>	complete a command or variable name
<code>M-?</code>	list possible completions
<code>RET</code>	enter the current line
<code>C-p</code>	move ‘up’ through the history list
<code>C-n</code>	move ‘down’ through the history list
<code>M-<</code>	move to the first line in the history
<code>M-></code>	move to the last line in the history
<code>C-r</code>	search backward in the history list
<code>C-s</code>	search forward in the history list

`history [-q] [N]` list *N* previous history lines, omitting history numbers if `-q`

`history -w [file]` write history to *file* (`~/.octave_hist` if no *file* argument)

`history -r [file]` read history from *file* (`~/.octave_hist` if no *file* argument)

`edit_history lines` edit and then run previous commands from the history list

`run_history lines` run previous commands from the history list

[*beg*] [*end*] Specify the first and last history commands to edit or run.

If *beg* is greater than *end*, reverse the list of commands before editing. If *end* is omitted, select commands from *beg* to the end of the history list. If both arguments are omitted, edit the previous item in the history list.

Shell Commands

<code>cd dir</code>	change working directory to <i>dir</i>
<code>pwd</code>	print working directory
<code>ls [options]</code>	print directory listing
<code>getenv (string)</code>	return value of named environment variable
<code>system (cmd)</code>	execute arbitrary shell command string

Matrices

Square brackets delimit literal matrices. Commas separate elements on the same row. Semicolons separate rows. Commas may be replaced by spaces, and semicolons may be replaced by one or more newlines. Elements of a matrix may be arbitrary expressions, provided that all the dimensions agree.

[<i>x</i> , <i>y</i> , ...]	enter a row vector
[<i>x</i> ; <i>y</i> ; ...]	enter a column vector
[<i>w</i> , <i>x</i> ; <i>y</i> , <i>z</i>]	enter a 2×2 matrix

Ranges

base : *limit*
base : *incr* : *limit*

Specify a range of values beginning with *base* with no elements greater than *limit*. If it is omitted, the default value of *incr* is 1. Negative increments are permitted.

Strings and Characters

A *string* consists of one or more characters. Characters may appear in either double or single quotes. Backslashes are used to escape special characters. The following table lists the escape sequences that can be used in strings.

<code>\\</code>	backslash
<code>\"</code>	double quote
<code>\'</code>	single quote
<code>\n</code>	newline
<code>\t</code>	tab

Index Expressions

var (idx) Access the *idx*th element of the vector *var*.
var (idx1, idx2, ...) Access the elements of *var* indexed by the scalar expressions *idx1*, *idx2*, ...

var (:) Access all elements of *var*.

var (: range) Access the elements of *var* indexed by the range *range*.

var (: :) Access all elements of *var*.

Global Variables

`global var1, var2, ...` Declare the global variables *var1*, *var2*, ... as global. This is useful for functions which need to access a global variable. The function must be declared `global` before the function body is executed.

Selected Environment Variables

`EDITOR` The editor to use for editing files.
`Inf`, `NaN` The values of infinity and not-a-number.
`LOADPATH` The list of directories to search for loadable modules.
`PAGER` The program to use for displaying the output of the `help` command.
`ans` The value of the last expression evaluated.
`eps` The machine epsilon.
`pi` The value of pi.
`realmax` The maximum real number.
`realmin` The minimum real number.

`automatic_reload` If set, Octave will automatically reload the current script if it has been modified.
`do_fortran_interface` If set, Octave will generate Fortran interface code for the current script.
`implicit_strconv` If set, Octave will convert strings to doubles implicitly.
`output_max_files` The maximum number of output files to keep.
`output_precision` The precision of the output.
`page_screen_size` The number of lines per page.
`prefer_column-major` If set, Octave will use column-major ordering for matrices.
`resize_on_resize` If set, Octave will resize the window when the size of the data changes.
`save_precision` The precision of the saved files.
`silent_function_errors` If set, Octave will not display error messages for function errors.
`warn_divide_by_zero` If set, Octave will warn about division by zero.

`commas_in_literals` If set, Octave will allow commas in literals.
`control_characters` The control characters to use.

`ignore_function_errors` If set, Octave will ignore errors in function calls.

`ok_to_lose_precision` If set, Octave will allow loss of precision.

`allow_complex_exponents` If set, Octave will allow complex exponents.

`prefer_zero_to_nan` If set, Octave will prefer zero to NaN.

Arithmetic and Increment Operators

<code>x + y</code>	addition
<code>x - y</code>	subtraction
<code>x * y</code>	matrix multiplication
<code>x .* y</code>	element by element multiplication
<code>x / y</code>	right division, conceptually equivalent to <code>(inverse (y') * x')</code>
<code>x ./ y</code>	element by element right division
<code>x \ y</code>	left division, conceptually equivalent to <code>inverse (x) * y</code>
<code>x.\ y</code>	element by element left division
<code>x ^ y</code>	power operator
<code>x .^ y</code>	element by element power operator
<code>- x</code>	negation
<code>+ x</code>	unary plus (a no-op)
<code>x '</code>	complex conjugate transpose
<code>x .' </code>	transpose
<code>++ x (-- x)</code>	increment (decrement) <i>x</i> , return <i>new</i> value
<code>x ++ (x --)</code>	increment (decrement) <i>x</i> , return <i>old</i> value

Assignment Expressions

<code>var = expr</code>	assign expression to variable
<code>var (idx) = expr</code>	assign expression to indexed variable

Comparison and Boolean Operators

These operators work on an element-by-element basis. Both arguments are always evaluated.

<code>x < y</code>	true if <i>x</i> is less than <i>y</i>
<code>x <= y</code>	true if <i>x</i> is less than or equal to <i>y</i>
<code>x == y</code>	true if <i>x</i> is greater than <i>y</i>
<code>x >= y</code>	true if <i>x</i> is greater than or equal to <i>y</i>
<code>x > y</code>	true if <i>x</i> is equal to <i>y</i>
<code>x != y</code>	true if <i>x</i> is not equal to <i>y</i>
<code>x & y</code>	true if both <i>x</i> and <i>y</i> are true
<code>x y</code>	true if at least one of <i>x</i> or <i>y</i> is true
<code>! bool</code>	true <i>bool</i> is false

Short-circuit Boolean Operators

Operators evaluate left-to-right, expecting scalar operands. Operands are only evaluated if necessary, stopping once overall truth value can be determined. Operands are converted to scalars by applying the `all` function.

<code>x && y</code>	true if both <i>x</i> and <i>y</i> are true
<code>x y</code>	true if at least one of <i>x</i> or <i>y</i> is true

Operator Precedence

Here is a table of the operators in Octave, in order of increasing precedence.

<code>;</code>	statement separators
<code>=</code>	assignment, groups left to right
<code> &&</code>	logical “or” and “and”
<code> &</code>	element-wise “or” and “and”
<code>< <= == >= > !=</code>	relational operators
<code>:</code>	colon
<code>+ -</code>	addition and subtraction
<code>* / \ .* ./ .\</code>	multiplication and division
<code>' .' </code>	transpose
<code>+ - ++ -- !</code>	unary minus, increment, logical “not”
<code>^ .^</code>	exponentiation

Statements

for *identifier* = *expr stmt-list endfor*
Execute *stmt-list* once for each column of *expr*. The variable *identifier* is set to the value of the current column during each iteration.

while (*condition*) *stmt-list endwhile*
Execute *stmt-list* while *condition* is true.

break exit innermost loop
continue go to beginning of innermost loop
return return to calling function

if (*condition*) *if-body* [**else** *else-body*] **endif**
Execute *if-body* if *condition* is true, otherwise execute *else-body*.

if (*condition*) *if-body* [**elseif** (*condition*) *elseif-body*] **endif**
Execute *if-body* if *condition* is true, otherwise execute the *elseif-body* corresponding to the first **elseif** condition that is true, otherwise execute *else-body*.
Any number of **elseif** clauses may appear in an **if** statement.

unwind_protect *body unwind_protect_cleanup cleanup end*
Execute *body*. Execute *cleanup* no matter how control exits *body*.

Defining Functions

function [*ret-list*] *function-name* [*(arg-list)*]
function-body
endfunction

ret-list may be a single identifier or a comma-separated list of identifiers delimited by square-brackets.
arg-list is a comma-separated list of identifiers and may be empty.

Basic Matrix Manipulations

<code>rows (a)</code>	return number of rows of <i>a</i>
<code>columns (a)</code>	return number of columns of <i>a</i>
<code>all (a)</code>	check if all elements of <i>a</i> nonzero
<code>any (a)</code>	check if any elements of <i>a</i> nonzero
<code>find (a)</code>	return indices of nonzero elements
<code>sort (a)</code>	order elements in each column of <i>a</i>
<code>sum (a)</code>	sum elements in columns of <i>a</i>
<code>prod (a)</code>	product of elements in columns of <i>a</i>
<code>min (args)</code>	find minimum values
<code>max (args)</code>	find maximum values
<code>rem (x, y)</code>	find remainder of <i>x/y</i>
<code>reshape (a, m, n)</code>	reformat <i>a</i> to be <i>m</i> by <i>n</i>

<code>diag (v, k)</code>	create diagonal matrices
<code>linspace (b, l, n)</code>	create vector of linearly-spaced elements
<code>logspace (b, l, n)</code>	create vector of log-spaced elements
<code>eye (n, m)</code>	create <i>n</i> by <i>m</i> identity matrix
<code>ones (n, m)</code>	create <i>n</i> by <i>m</i> matrix of ones
<code>zeros (n, m)</code>	create <i>n</i> by <i>m</i> matrix of zeros
<code>rand (n, m)</code>	create <i>n</i> by <i>m</i> matrix of random values

Linear Algebra

<code>chol (a)</code>
<code>det (a)</code>
<code>eig (a)</code>
<code>expm (a)</code>
<code>hess (a)</code>
<code>inverse (a)</code>
<code>norm (a, p)</code>
<code>pinv (a)</code>
<code>qr (a)</code>
<code>rank (a)</code>
<code>schur (a)</code>
<code>svd (a)</code>
<code>syl (a, b, c)</code>

Equations

`*fsolve`
`*lsode`
`*dassl`
`*quad`

`perror (nm,`

`* See the on-`
`arguments for`

Signal Processing

<code>fft (a)</code>
<code>ifft (a)</code>
<code>freqz (args)</code>
<code>sinc (x)</code>

Image Processing

<code>colormap (m)</code>
<code>gray2ind (i,</code>
<code>image (img,</code>
<code>imagesc (img,</code>
<code>imshow (img,</code>
<code>imshow (i, n,</code>
<code>imshow (r, g,</code>
<code>ind2gray (im</code>
<code>ind2rgb (img</code>
<code>loadimage (f</code>
<code>rgb2ind (r,</code>
<code>saveimage (f</code>

Sets

<code>create_set (</code>
<code>complement (</code>
<code>intersection</code>
<code>union (a, b)</code>

Strings

<code>strcmp (s, t)</code>
<code>strcat (s, t,</code>

C-style Input and Output

<code>fopen</code> (<i>name</i> , <i>mode</i>)	open file <i>name</i>
<code>fclose</code> (<i>file</i>)	close <i>file</i>
<code>printf</code> (<i>fmt</i> , ...)	formatted output to <code>stdout</code>
<code>fprintf</code> (<i>file</i> , <i>fmt</i> , ...)	formatted output to <i>file</i>
<code>sprintf</code> (<i>fmt</i> , ...)	formatted output to string
<code>scanf</code> (<i>fmt</i>)	formatted input from <code>stdin</code>
<code>fscanf</code> (<i>file</i> , <i>fmt</i>)	formatted input from <i>file</i>
<code>sscanf</code> (<i>str</i> , <i>fmt</i>)	formatted input from <i>string</i>
<code>fgets</code> (<i>file</i> , <i>len</i>)	read <i>len</i> characters from <i>file</i>
<code>fflush</code> (<i>file</i>)	flush pending output to <i>file</i>
<code>ftell</code> (<i>file</i>)	return file pointer position
<code>frewind</code> (<i>file</i>)	move file pointer to beginning
<code>freport</code>	print a info for open files
<code>fread</code> (<i>file</i> , <i>size</i> , <i>prec</i>)	read binary data files
<code>fwrite</code> (<i>file</i> , <i>size</i> , <i>prec</i>)	write binary data files
<code>feof</code> (<i>file</i>)	determine if pointer is at EOF

A file may be referenced either by name or by the number returned from `fopen`. Three files are preconnected when Octave starts: `stdin`, `stdout`, and `stderr`.

Other Input and Output functions

<code>save</code> <i>file</i> <i>var</i> ...	save variables in <i>file</i>
<code>load</code> <i>file</i>	load variables from <i>file</i>
<code>disp</code> (<i>var</i>)	display value of <i>var</i> to screen

Miscellaneous Functions

<code>eval</code> (<i>str</i>)	evaluate <i>str</i> as a command
<code>feval</code> (<i>str</i> , ...)	evaluate function named by <i>str</i> , passing remaining args to called function

<code>error</code> (<i>message</i>)	print message and return to top level
---------------------------------------	---------------------------------------

<code>clear</code> <i>pattern</i>	clear variables matching pattern
<code>exist</code> (<i>str</i>)	check existence of variable or function
<code>who</code>	list current variables

Polynomials

<code>compan</code> (<i>p</i>)	companion matrix
<code>conv</code> (<i>a</i> , <i>b</i>)	convolution
<code>deconv</code> (<i>a</i> , <i>b</i>)	deconvolve two vectors
<code>poly</code> (<i>a</i>)	create polynomial from a matrix
<code>polyderiv</code> (<i>p</i>)	derivative of polynomial
<code>polyreduce</code> (<i>p</i>)	integral of polynomial
<code>polyval</code> (<i>p</i> , <i>x</i>)	value of polynomial at <i>x</i>
<code>polyvalm</code> (<i>p</i> , <i>x</i>)	value of polynomial at <i>x</i>
<code>roots</code> (<i>p</i>)	polynomial roots
<code>residue</code> (<i>a</i> , <i>b</i>)	partial fraction expansion of ratio <i>a/b</i>

Statistics

<code>corrcoef</code> (<i>x</i> , <i>y</i>)	correlation coefficient
<code>cov</code> (<i>x</i> , <i>y</i>)	covariance
<code>mean</code> (<i>a</i>)	mean value
<code>median</code> (<i>a</i>)	median value
<code>std</code> (<i>a</i>)	standard deviation
<code>var</code> (<i>a</i>)	variance

Basic Plotting

<code>gplot</code> [<i>ranges</i>] <i>expr</i> [<i>using</i>] [<i>title</i>] [<i>style</i>]	2D plotting
<code>gsplot</code> [<i>ranges</i>] <i>expr</i> [<i>using</i>] [<i>title</i>] [<i>style</i>]	3D plotting
<i>ranges</i>	specify data ranges
<i>expr</i>	expression to plot
<i>using</i>	specify columns to plot
<i>title</i>	specify line title for legend
<i>style</i>	specify line style

If *ranges* are supplied, they must come before the expression to plot. The *using*, *title*, and *style* options may appear in any order after *expr*. Multiple expressions may be plotted with a single command by separating them with commas.

<code>set</code> <i>options</i>	set plotting options
<code>show</code> <i>options</i>	show plotting options
<code>replot</code>	redisplay current plot
<code>closeplot</code>	close stream to <code>gnuplot</code> process
<code>purge_tmp_files</code>	clean up temporary plotting files
<code>automatic_replot</code>	built-in variable

Other Plotting Functions

<code>plot</code> (<i>args</i>)	2D plot with linear axes
<code>semilogx</code> (<i>args</i>)	2D plot with logarithmic x-axis
<code>semilogy</code> (<i>args</i>)	2D plot with logarithmic y-axis
<code>loglog</code> (<i>args</i>)	2D plot with logarithmic axes
<code>bar</code> (<i>args</i>)	plot bar charts
<code>stairs</code> (<i>x</i> , <i>y</i>)	plot stairsteps
<code>hist</code> (<i>y</i> , <i>x</i>)	plot histograms

<code>title</code> (<i>string</i>)	set plot title
--------------------------------------	----------------

<code>axis</code> (<i>limits</i>)	set axis ranges
<code>xlabel</code> (<i>string</i>)	set x-axis label
<code>ylabel</code> (<i>string</i>)	set y-axis label
<code>grid</code> [<i>on</i> <i>off</i>]	set grid state
<code>hold</code> [<i>on</i> <i>off</i>]	set hold state
<code>ishold</code>	return 1 if hold is on, 0 otherwise

<code>mesh</code> (<i>x</i> , <i>y</i> , <i>z</i>)	plot 3D surface
<code>meshdom</code> (<i>x</i> , <i>y</i>)	create mesh coordinate matrices

Edition 1.1for Octave Version 1.1.1. Copyright 1996, John W. Eaton (jwe@che.utexas.edu). The author assumes no responsibility for any errors on this card.

This card may be freely distributed under the terms of the GNU General Public License.

T_EX Macros for this card by Roland Pesch (pesch@cygnus.com), originally for the GDB reference card

Octave itself is free software; you are welcome to distribute copies of it under the terms of the GNU General Public License. There is absolutely no warranty for Octave.