

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
Copyright 1996, 1997 John W. Eaton Permissions on back

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 Common Escape Sequences

A *string constant* consists of a sequence of characters enclosed in either double-quote or single-quote marks.

<code>\\</code>	a literal backslash
<code>\"</code>	a literal double-quote character
<code>\'</code>	a literal single-quote character
<code>\n</code>	newline, ASCII code 10
<code>\t</code>	horizontal tab, ASCII code 9

Index Expressions

<code>var (idx)</code>	select elements of a vector
<code>var (idx1, idx2)</code>	select elements of a matrix
<code>scalar</code>	select row (column) corresponding to <i>scalar</i>
<code>vector</code>	select rows (columns) corresponding to the elements of <i>vector</i>
<code>range</code>	select rows (columns) corresponding to the elements of <i>range</i>
<code>:</code>	select all rows (columns)

Global Variables

`global var1 ...` Declare variables global.
Global variables may be accessed inside the body of a function without having to be passed in the function parameter list provided they are also declared global within the function.

Selected Built-in Variables

<code>EDITOR</code>	editor to use with <code>edit_history</code>
<code>Inf</code> , <code>NaN</code>	IEEE infinity, NaN
<code>LOADPATH</code>	path to search for function files
<code>PAGER</code>	program to use to paginate output
<code>ans</code>	last result not explicitly assigned
<code>eps</code>	machine precision
<code>pi</code>	π
<code>realmax</code>	maximum representable value
<code>realmin</code>	minimum representable value

<code>automatic_replot</code>	automatically redraw plots
<code>do_fortran_indexing</code>	Fortran-style indexing of matrices
<code>implicit_str_to_num_ok</code>	allow strings to become numbers
<code>output_max_field_width</code>	maximum numeric field width
<code>output_precision</code>	min significant figures displayed
<code>page_screen_output</code>	control whether output is paged
<code>prefer_column_vectors</code>	create column vectors by default
<code>resize_on_range_error</code>	automatic resizing of matrices
<code>save_precision</code>	digits stored by <code>save</code> command
<code>silent_functions</code>	suppress output from functions
<code>warn_divide_by_zero</code>	suppress divide by zero errors

`commas_in_literal_matrix`
control handling of spaces in matrices

`ignore_function_time_stamp`
ignore changes in function files during session

`ok_to_lose_imaginary_part`
allow complex to real conversion

`prefer_zero_one_indexing`
if ambiguous, prefer 0-1 style indexing

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</code> (<code>-- x</code>)	increment (decrement) <i>x</i> , return <i>new</i> value
<code>x ++</code> (<code>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 equal to <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 greater than <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 if <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

rows (*a*) return number of rows of *a*

columns (*a*) return number of columns of *a*

all (*a*) check if all elements of *a* nonzero

any (*a*) check if any elements of *a* nonzero

find (*a*) return indices of nonzero elements

sort (*a*) order elements in each column of *a*

sum (*a*) sum elements in columns of *a*

prod (*a*) product of elements in columns of *a*

min (*args*) find minimum values

max (*args*) find maximum values

rem (*x*, *y*) find remainder of *x/y*

reshape (*a*, *m*, *n*) reformat *a* to be *m* by *n*

diag (*v*, *k*) create diagonal matrices

linspace (*b*, *l*, *n*) create vector of linearly-spaced elements

logspace (*b*, *l*, *n*) create vector of log-spaced elements

eye (*n*, *m*) create *n* by *m* identity matrix

ones (*n*, *m*) create *n* by *m* matrix of ones

zeros (*n*, *m*) create *n* by *m* matrix of zeros

rand (*n*, *m*) create *n* by *m* matrix of random values

Linear Algebra

chol (*a*) Cholesky factorization

det (*a*) compute the determinant of a matrix

eig (*a*) eigenvalues and eigenvectors

expm (*a*) compute the exponential of a matrix

hess (*a*) compute Hessenberg decomposition

inverse (*a*) invert a square matrix

norm (*a*, *p*) compute the *p*-norm of a matrix

pinv (*a*) compute pseudoinverse of *a*

qr (*a*) compute the QR factorization of a matrix

rank (*a*) matrix rank

schur (*a*) Schur decomposition of a matrix

svd (*a*) singular value decomposition

syl (*a*, *b*, *c*) solve the Sylvester equation

Equations, ODEs, DAEs, Quadrature

***fsolve** solve nonlinear algebraic equations

***lsode** integrate nonlinear ODEs

***dassl** integrate nonlinear DAEs

***quad** integrate nonlinear functions

perror (*nm*, *code*) for functions that return numeric codes, print error message for named function and given error code

* See the on-line or printed manual for the complete list of arguments for these functions.

Signal Processing

fft (*a*) Fast Fourier Transform using FFTPACK

ifft (*a*) inverse FFT using FFTPACK

freqz (*args*) FIR filter frequency response

sinc (*x*) returns $\sin(\pi\text{ }x)/(\pi\text{ }x)$

Image Processing

colormap (*map*) set the current colormap

gray2ind (*i*, *n*) convert gray scale to Octave image

image (*img*, *zoom*) display an Octave image matrix

imagesc (*img*, *zoom*) display scaled matrix as image

imshow (*img*, *map*) display Octave image

imshow (*i*, *n*) display gray scale image

imshow (*r*, *g*, *b*) display RGB image

ind2gray (*img*, *map*) convert Octave image to gray scale

ind2rgb (*img*, *map*) convert indexed image to RGB

loadimage (*file*) load an image file

rgb2ind (*r*, *g*, *b*) convert RGB to Octave image

saveimage (*file*, *img*, *fmt*, *map*) save a matrix to *file*

Sets

create_set (*a*, *b*) create row vector of unique values

complement (*a*, *b*) elements of *b* not in *a*

intersection (*a*, *b*) intersection of sets *a* and *b*

union (*a*, *b*) union of sets *a* and *b*

Strings

strcmp (*s*, *t*) compare strings

strcat (*s*, *t*, ...) concatenate strings

C-style Input and Output

fopen (*name*, *mode*) open file *name*

fclose (*file*) close *file*

printf (*fmt*, ...) formatted output to **stdout**

fprintf (*file*, *fmt*, ...) formatted output to *file*

sprintf (*fmt*, ...) formatted output to string

scanf (*fmt*) formatted input from **stdin**

fscanf (*file*, *fmt*) formatted input from *file*

sscanf (*str*, *fmt*) formatted input from *string*

fgets (*file*, *len*) read *len* characters from *file*

fflush (*file*) flush pending output to *file*

ftell (*file*) return file pointer position

frewind (*file*) move file pointer to beginning

freport print a info for open files

fread (*file*, *size*, *prec*) read binary data files

fwrite (*file*, *size*, *prec*) write binary data files

feof (*file*) 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

save *file* *var* ... save variables in *file*

load *file* load variables from *file*

disp (*var*) display value of *var* to screen

Miscellaneous Functions

eval (*str*) evaluate *str* as a command

feval (*str*, ...) evaluate function named by *str*, passing remaining args to called function

error (*message*) print message and return to top level

clear *pattern* clear variables matching pattern

exist (*str*) check existence of variable or function

who list current variables

Polynomials

compan (*p*) companion matrix

conv (*a*, *b*) convolution

deconv (*a*, *b*) deconvolve two vectors

poly (*a*) create polynomial from a matrix

polyderiv (*p*) derivative of polynomial

polyreduce (*p*) integral of polynomial

polyval (*p*, *x*) value of polynomial at *x*

polyvalm (*p*, *x*) value of polynomial at *x*

roots (*p*) polynomial roots

residue (*a*, *b*) partial fraction expansion of ratio *a/b*

Statistics

corrcoef (*x*, *y*) correlation coefficient

cov (*x*, *y*) covariance

mean (*a*) mean value

median (*a*) median value

std (*a*) standard deviation

var (*a*) variance

Basic Plotting

gplot [*ranges*] *expr* [*using*] [*title*] [*style*] 2D plotting

gsplot [*ranges*] *expr* [*using*] [*title*] [*style*] 3D plotting

ranges specify data ranges

expr expression to plot

using specify columns to plot

title specify line title for legend

style 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.

set *options* set plotting options

show *options* show plotting options

replot redisplay current plot

closeplot close stream to **gnuplot** process

purge_tmp_files clean up temporary plotting files

automatic_replot built-in variable

Other Plotting Functions

plot (*args*) 2D plot with linear axes

semilogx (*args*) 2D plot with logarithmic x-axis

semilogy (*args*) 2D plot with logarithmic y-axis

loglog (*args*) 2D plot with logarithmic axes

bar (*args*) plot bar charts

stairs (*x*, *y*) plot stairsteps

hist (*y*, *x*) plot histograms

title (*string*) set plot title

axis (*limits*) set axis ranges

xlabel (*string*) set x-axis label

ylabel (*string*) set y-axis label

grid [**on**|**off**] set grid state

hold [**on**|**off**] set hold state

ishold return 1 if hold is on, 0 otherwise

mesh (*x*, *y*, *z*) plot 3D surface

meshdom (*x*, *y*) create mesh coordinate matrices

Edition 1.1.for 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.