

Starting and Stopping

<code>octave</code> [<code>--gui</code>]	start Octave CLI/GUI session
<code>octave</code> <i>file</i>	run Octave commands in <i>file</i>
<code>octave</code> <code>--eval</code> <i>code</i>	evaluate <i>code</i> using Octave
<code>octave</code> <code>--help</code>	describe command line options
<code>quit</code> or <code>exit</code>	exit Octave
<code>Ctrl-C</code>	terminate current command and return to top-level prompt

Getting Help

<code>help</code> <i>command</i>	briefly describe <i>command</i>
<code>doc</code>	use Info to browse Octave manual
<code>doc</code> <i>command</i>	search for <i>command</i> in Octave manual
<code>lookfor</code> <i>str</i>	search for <i>command</i> based on <i>str</i>

Command Completion and History

<code>TAB</code>	complete a command or variable name
<code>Alt-?</code>	list possible completions
<code>Ctrl-r</code> <code>Ctrl-s</code>	search command history

Directory and Path Commands

<code>cd</code> <i>dir</i>	change working directory to <i>dir</i>
<code>pwd</code>	print working directory
<code>ls</code> [<i>options</i>]	print directory listing
<code>what</code>	list <code>.m/.mat</code> files in the current directory
<code>path</code>	search path for Octave functions
<code>pathdef</code>	default search path
<code>addpath</code> (<i>dir</i>)	add a directory to the path
<code>getenv</code> (<i>var</i>)	value of environment variable

Package Management

Add-on packages are independent of core Octave, listed at <https://packages.octave.org/>

<code>pkg install</code> <code>-forge</code> <i>pkg</i>	download and install <i>pkg</i>
<code>pkg install</code> <i>file.tar.gz</i>	install pre-downloaded package file
<code>pkg list</code>	show installed packages
<code>pkg load</code> / <code>pkg unload</code>	load/unload installed package
<code>statistics</code> <code>optimization</code>	various common packages
<code>control</code> <code>signal</code> <code>image</code>	
<code>symbolic</code> etc.	

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 newlines. Elements of a matrix may be arbitrary expressions, assuming all the dimensions agree.

<code>[</code> <i>x</i> , <i>y</i> , ... <code>]</code>	enter a row vector
<code>[</code> <i>x</i> ; <i>y</i> ; ... <code>]</code>	enter a column vector
<code>[</code> <i>w</i> , <i>x</i> ; <i>y</i> , <i>z</i> <code>]</code>	enter a 2×2 matrix
<code>rows</code> <code>columns</code>	number of rows/columns of matrix
<code>zeros</code> <code>ones</code>	create matrix of zeros/ones
<code>eye</code> <code>diag</code>	create identity/diagonal matrix
<code>rand</code> <code>randi</code> <code>randn</code>	create matrix of random values
<code>sparse</code> <code>spalloc</code>	create a sparse matrix
<code>all</code>	true if all elements nonzero

<code>any</code>	true if at least one element nonzero
<code>nnz</code>	number of nonzero elements

Multi-dimensional Arrays

<code>ndims</code>	number of dimensions
<code>reshape</code> <code>squeeze</code>	change array shape
<code>resize</code>	change array shape, lossy
<code>cat</code>	join arrays along a given dimension
<code>permute</code> <code>ipermute</code>	like N-dimensional transpose
<code>shiftdim</code>	
<code>circshift</code>	cyclically shift array elements
<code>meshgrid</code>	matrices useful for vectorization

Ranges

Create sequences of real numbers as row vectors.

<i>base</i> : <i>limit</i>	
<i>base</i> : <i>incr</i> : <i>limit</i>	
<i>incr</i> == 1 if not specified. Negative ranges allowed.	

Numeric Types and Values

Integers saturate in Octave. They do not roll over.

<code>int8</code> <code>int16</code> <code>int32</code> <code>int64</code>	signed integers
<code>uint8</code> <code>uint16</code> <code>uint32</code>	unsigned integers
<code>uint64</code>	
<code>single</code> <code>double</code>	32-bit/64-bit IEEE floating point
<code>intmin</code> <code>intmax</code> <code>flintmax</code>	integer limits of given type
<code>realmin</code> <code>realmax</code>	floating point limits of given type
<code>inf</code> <code>nan</code> <code>NA</code>	IEEE infinity, NaN, missing value
<code>eps</code>	machine precision
<code>pi</code> <code>e</code>	3.14159..., 2.71828...
<code>i</code> <code>j</code>	$\sqrt{-1}$

Strings

A *string constant* consists of a sequence of characters enclosed in either double-quote or single-quote marks. Strings in double-quotes allow the use of the escape sequences below.

<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
<code>sprintf</code> <code>sscanf</code>	formatted IO to/from string
<code>strcmp</code>	compare strings
<code>strcat</code>	join strings
<code>strfind</code> <code>regex</code>	find matching patterns
<code>strrep</code> <code>regexprep</code>	find and replace patterns

Index Expressions

<i>var</i> (<i>idx</i>)	select elements of a vector
<i>var</i> (<i>idx1</i> , <i>idx2</i>)	select elements of a matrix
<i>var</i> ([1 3], :)	rows 1 and 3
<i>var</i> (:, [2 end])	the second and last columns
<i>var</i> (1:2:end, 2:2:end)	get odd rows and even columns
<i>var</i> 1(<i>var</i> 2 == 0)	elements of <i>var</i> 1 corresponding to zero elements of <i>var</i> 2
<i>var</i> (:)	all elements as a column vector

Cells, Structures, and Classdefs

<i>var</i> { <i>idx</i> }	= ...	set an element of a cell array
<code>cellfun</code> (<i>f</i> , <i>c</i>)		apply a function to elements of cell array
<i>var</i> . <i>field</i> = ...		set a field of a structure
<code>fieldnames</code> (<i>s</i>)		returns the fields of a structure
<code>structfun</code> (<i>f</i> , <i>s</i>)		apply a function to fields of structure
<code>classdef</code>		define new classes for OOP

Assignment Expressions

<i>var</i> = <i>expr</i>	assign value to variable
<i>var</i> (<i>idx</i>) = <i>expr</i>	only the indexed elements are changed
<i>var</i> (<i>idx</i>) = []	delete the indexed elements

Arithmetic Operators

If two operands are of different sizes, scalars and singleton dimensions are automatically expanded. Non-singleton dimensions need to match.

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

Comparison and Boolean Operators

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

<code><</code> <code><=</code> <code>==</code> <code>>=</code> <code>></code>	relational operators
<code>!=</code> <code>~=</code>	not equal to
<code>&</code>	logical AND
<code> </code>	logical OR
<code>!</code> <code>~</code>	logical NOT

Short-circuit Boolean Operators

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

<i>x</i> && <i>y</i>	logical AND
<i>x</i> <i>y</i>	logical OR

Operator Precedence

Table of Octave operators, in order of **decreasing** precedence.

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

General programming

`endfor`, `endwhile`, `endif` etc. can all be replaced by `end`.

<code>for</code> <code>x</code> = 1:10	for loop
<code>endfor</code>	
<code>while</code> (<code>x</code> <= 10)	while loop
<code>endwhile</code>	
<code>do</code>	do-until loop
<code>until</code> (<code>x</code> > 10)	
<code>if</code> (<code>x</code> < 5)	if-then-else
<code>elseif</code> (<code>x</code> < 6)	
<code>else</code>	
<code>endif</code>	
<code>switch</code> (<code>tf</code>)	switch-case
<code>case</code> "true"	
<code>case</code> "false"	
<code>otherwise</code>	
<code>endswitch</code>	
<code>break</code>	exit innermost loop
<code>continue</code>	go to start of innermost loop
<code>return</code>	jump back from function to caller
<code>try</code>	cleanup only on exception
<code>catch</code>	
<code>unwind_protect</code>	cleanup always
<code>unwind_protect_cleanup</code>	

Functions

<code>function</code> [<i>ret-list</i> =] <i>function-name</i> [(<i>arg-list</i>)]
<i>function-body</i>
<code>endfunction</code>

ret-list may be a single identifier or a comma-separated list of identifiers enclosed by square brackets.

arg-list is a comma-separated list of identifiers and may be empty.

Function Handles and Evaluation

<code>@func</code>	create a function handle to <i>func</i>
<code>@(vars)</code> <i>expr</i>	define an anonymous function
<code>str2func</code> <code>func2str</code>	convert function to/from string

functions (<i>handle</i>)	Return information about a function handle
<i>f</i> (<i>args</i>)	Evaluate a function handle <i>f</i>
feval	Evaluate a function handle or string
eval (<i>str</i>)	evaluate <i>str</i> as a command
system (<i>cmd</i>)	execute arbitrary shell command string

Anonymous function handles make a copy of the variables in the current workspace at the time of creation.

Global and Persistent Variables

global *var* = ... declare & initialize global variable
persistent *var* = ... persistent/static variable
Global variables may be accessed inside the body of a function without having to be passed in the function parameter list provided that they are declared global when used.

Common Functions

disp	display value of variable
printf	formatted output to stdout
input scanf	input from stdin
who whos	list current variables
clear <i>pattern</i>	clear variables matching pattern
exist	check existence of identifier
find	return indices of nonzero elements
sort	return a sorted array
unique	discard duplicate elements
sortrows	sort whole rows in numerical or lexicographic order
sum prod	sum or product
mod rem	remainder functions
min max range	basic statistics
mean median std	

Error Handling, Debugging, Profiling

error (<i>message</i>)	print message and return to top level
warning (<i>message</i>)	print a warning message
debug	guide to all debugging commands
profile	start/stop/clear/resume profiling
profshow	show the results of profiling
profexplore	

File I/O, Loading, Saving

save load	save/load variables to/from file
save -binary	save in binary format (faster)
dlmread dlmwrite	read/write delimited data
csvread csvwrite	read/write CSV files
xlsread xlswrite	read/write XLS spreadsheets

fopen fclose	open/close files
fprintf fscanf	formatted file I/O
textscan	
fflush	flush pending output

Math Functions

Run `doc <function>` to find related functions.

cov corrcoef	covariance, correlation coefficient
tan tanh atan2	trig and hyperbolic functions
cross curl del2	vector algebra functions

det inv	determinant matrix inverse
eig	eigenvalues and eigenvectors
norm	vector norm, matrix norm
rank	matrix rank
qr	QR factorization
chol	Cholesky factorization
svd	singular value decomposition

fsolve	solve nonlinear algebraic equations
lsode ode45	integrate nonlinear ODEs
dassl	integrate nonlinear DAEs
integral	integrate nonlinear functions

union	set union
intersection	set intersection
setdiff	set difference

roots	polynomial roots
poly	matrix characteristic polynomial
polyder polyint	polynomial derivative or integral
polyfit polyval	polynomial fitting and evaluation
residue	partial fraction expansion
legendre bessel	special functions

conv conv2	convolution, polynomial multiplication
deconv	deconvolution, polynomial division

fft fft2 ifft (<i>a</i>)	FFT / inverse FFT
freqz	FIR filter frequency response
filter	filter by transfer function

Plotting and Graphics

plot plot3	2D / 3D plot with linear axes
line	2D or 3D line
patch fill	2D patch, optionally colored
semilogx semilogy	logarithmic axes
loglog	

bar hist	bar chart, histogram
stairs stem	stairsteps and stem graphs
contour	contour plot
mesh trimesh surf	plot 3D surfaces

figure	new figure
hold on	add to existing figure
title	set plot title
axis	set axis range and aspect
xlabel ylabel zlabel	set axis labels
text	add text to a plot
grid legend	draw grid or legend

image imagesc spy	display matrix as image
imwrite saveas print	save figure or image
imread	load an image
colormap	get or set colormap

Quick reference for Octave 8.0.0. Copyright 1996-2023 The Octave Project Developers. The authors assume no responsibility for any errors on this card. This card may be freely distributed under the terms of the GNU General Public License.

Octave license and copyright: <https://octave.org/copyright/>

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