Octave Quick Reference

Octave 8.0.0

Starting and Stopping

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

Getting Help

`help` *command*	briefly describe *command*
`doc`	use Info to browse Octave manual
`doc` *command*	search for *command* in Octave manual
`lookfor` *str*	search for *command* based on *str*

Command Completion and History

`TAB`	complete a command or variable name
`Alt-?`	list possible completions
`Ctrl-r Ctrl-s`	search command history

Directory and Path Commands

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

Package Management

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

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

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.

`[` *x*`,` *y*`, ... ]`	enter a row vector
`[` *x*`;` *y*`; ... ]`	enter a column vector
`[` *w*`,` *x*`;` *y*`,` *z* `]`	enter a 2×2 matrix
`rows columns`	number of rows/columns of matrix
`zeros ones`	create matrix of zeros/ones
`eye diag`	create identity/diagonal matrix
`rand randi randn`	create matrix of random values
`sparse spalloc`	create a sparse matrix
`all`	true if all elements nonzero
`any`	true if at least one element nonzero
`nnz`	number of nonzero elements

Multi-dimensional Arrays

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

Ranges

Create sequences of real numbers as row vectors.

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

incr == 1 if not specified. Negative ranges allowed.

Numeric Types and Values

Integers saturate in Octave. They do not roll over.

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

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.

`\\`	a literal backslash
`\"`	a literal double-quote character
`\'`	a literal single-quote character
`\n`	newline, ASCII code 10
`\t`	horizontal tab, ASCII code 9
`sprintf sscanf`	formatted IO to/from string
`strcmp`	compare strings
`strcat`	join strings
`strfind regexp`	find matching patterns
`strrep regexprep`	find and replace patterns

Index Expressions

var`(`*idx*`)`	select elements of a vector
var`(`*idx1*`,` *idx2*`)`	select elements of a matrix
var`([1 3], :)`	rows 1 and 3
var`(:, [2 end])`	the second and last columns
var`(1:2:end, 2:2:end)`	get odd rows and even columns
var1`(`*var2* `== 0)`	elements of *var1* corresponding to zero elements of *var2*
var`(:)`	all elements as a column vector

Cells, Structures, and Classdefs

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

Assignment Expressions

var `=` *expr*	assign value to variable
var`(`*idx*`) =` *expr*	only the indexed elements are changed
var`(`*idx*`) = []`	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.

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

Comparison and Boolean Operators

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

`< <= == >= >`	relational operators
`!= ~=`	not equal to
`&`	logical AND
`|`	logical OR
`! ~`	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`.

x `&&` *y*	logical AND
x `||` *y*	logical OR

Operator Precedence

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

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

General programming

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

`for x = 1:10` `endfor`	for loop
`while (x <= 10)` `endwhile`	while loop
`do` `until (x > 10)`	do-until loop

`if (x < 5)`
`elseif (x < 6)`
`else`
`endif`
	if-then-else

`switch (tf)`
`case "true"`
`case "false"`
`otherwise`
`endswitch`
	switch-case

`break`	exit innermost loop
`continue`	go to start of innermost loop
`return`	jump back from function to caller
`try` `catch`	cleanup only on exception
`unwind_protect` `unwind_protect_cleanup`	cleanup always

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 enclosed by square brackets.

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

Function Handles and Evaluation

`@`*func*	create a function handle to *func*
`@(`*vars*`)` *expr*	define an anonymous function
`str2func func2str`	convert function to/from string

`functions` (*handle*)	Return information about a function handle
f (*args*)	Evaluate a function handle *f*
`feval`	Evaluate a function handle or string
`eval` (*str*)	evaluate *str* as a command
`system` (*cmd*)	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` *pattern*	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` (*message*)	print message and return to top level
`warning` (*message*)	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(`*a*`)`	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 loglog`	logarithmic axes
`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

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