

# Octave Quick Reference

Octave 8 or later

Copyright © 1996-2025 The Octave Project Developers

## Starting and Stopping

<code>octave [-gui]</code>	start Octave CLI/GUI session
<code>octave file</code>	run Octave commands in <i>file</i>
<code>octave --eval code</code>	evaluate <i>code</i> using Octave
<code>octave --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 command</code>	briefly describe <i>command</i>
<code>doc</code>	browse Octave manual
<code>doc command</code>	search for <i>command</i> in Octave manual
<code>lookfor str</code>	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 Ctrl-s</code>	search command history

## Directory and Path 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>what</code>	list .m/.mat files in the current directory
<code>path</code>	search path for Octave functions
<code>pathdef</code>	default search path
<code>addpath (dir)</code>	add directory to search path
<code>getenv (var)</code>	value of environment variable

## Package Management

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

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

## 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>[ x, y, ... ]</code>	enter a row vector
<code>[ x; y; ... ]</code>	enter a column vector
<code>[ w, x; y, z ]</code>	enter a 2×2 matrix
<code>rows columns</code>	number of rows/columns of matrix
<code>zeros ones</code>	create matrix of zeros/ones
<code>eye diag</code>	create identity/diagonal matrix
<code>rand randi randn</code>	create matrix of random values
<code>sparse 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 squeeze</code>	change array shape
<code>resize</code>	change array shape, lossy
<code>cat</code>	join arrays along a given dimension
<code>permute 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.

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

## Numeric Types and Values

Integers saturate in Octave. They do not roll over.

<code>int8 int16 int32 int64</code>	signed integers
<code>uint8 uint16 uint32 uint64</code>	unsigned integers
<code>single double</code>	32-bit/64-bit IEEE floating point
<code>intmin intmax flintmax</code>	integer limits of given type
<code>realmin realmax</code>	floating point limits of given type
<code>inf nan NA</code>	IEEE infinity, NaN, missing value
<code>eps</code>	machine precision
<code>pi e</code>	3.14159..., 2.71828...
<code>i 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 sscanf</code>	formatted IO to/from string
<code>strcmp</code>	compare strings
<code>strcat</code>	join strings
<code>strfind regexp</code>	find matching patterns
<code>strrep regexprep</code>	find and replace patterns

## Index Expressions

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

## Cells, Structures, and Classdefs

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

## Assignment Expressions

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

<code>x + y, x - y</code>	addition, subtraction
<code>x * y</code>	matrix multiplication
<code>x .* y</code>	element-by-element multiplication
<code>x / y</code>	right division, conceptually equivalent to <b>(inverse (y') * x')</b>
<code>x ./ y</code>	element-by-element right division
<code>x \ y</code>	left division, conceptually equivalent to <b>inverse (x) * y</b>
<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>+= -= *= .*= ./=</code>	in-place equivalents of the above operators
<code>./= \= .\= ^= .^=</code>	
<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, return <i>new</i> value
<code>x++ 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>&lt; &lt;= == &gt; &gt;=</code>	relational operators
<code>!= ~=</code>	not equal to
<code>&amp;</code>	logical AND
<code> </code>	logical OR
<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**.

<code>x &amp;&amp; y</code>	logical AND
<code>x    y</code>	logical OR

## Operator Precedence

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

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

## General programming

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

<code>for x = 1:10</code>	for loop
<code>endfor</code>	
<code>while (x &lt;= 10)</code>	while loop
<code>endwhile</code>	
<code>do</code>	do-until loop
<code>until (x &gt; 10)</code>	
<code>if (x &lt; 5)</code>	if-then-else
<code>elseif (x &lt; 6)</code>	
<code>else</code>	
<code>endif</code>	
<code>switch (tf)</code>	switch-case
<code>  case "true"</code>	
<code>  case "false"</code>	
<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

```
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 mean basic statistics
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, smaller)
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

type 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 or 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/integral
polyfit polyval polynomial fitting/evaluation
residue        partial fraction expansion
legendre bessell special functions
```

```
conv conv2     convolution, polynomial multiplication
deconv         deconvolution, polynomial division
```

```
fft fft2 ifft  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    staircase and stem plots
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 or later. Copyright © 1996-2025 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