

# Emacs Lisp

The homepage for GNU Emacs is at <http://www.gnu.org/software/emacs/>.

For information on using Emacs, refer to the [Emacs Manual](#).

To view this manual in other formats, click [here](#).

This is the GNU Emacs Lisp Reference Manual corresponding to Emacs version 25.2.

<a href="#">Introduction</a>	Introduction and conventions used.
<a href="#">Lisp Data Types</a>	Data types of objects in Emacs Lisp.
<a href="#">Numbers</a>	Numbers and arithmetic functions.
<a href="#">Strings and Characters</a>	Strings, and functions that work on them.
<a href="#">Lists</a>	Lists, cons cells, and related functions.
<a href="#">Sequences Arrays Vectors</a>	Lists, strings and vectors are called sequences. Certain functions act on any kind of sequence. The description of vectors is here as well.
<a href="#">Hash Tables</a>	Very fast lookup-tables.
<a href="#">Symbols</a>	Symbols represent names, uniquely.
<a href="#">Evaluation</a>	How Lisp expressions are evaluated.
<a href="#">Control Structures</a>	Conditionals, loops, nonlocal exits.
<a href="#">Variables</a>	Using symbols in programs to stand for values.
<a href="#">Functions</a>	A function is a Lisp program that can be invoked from other functions.
<a href="#">Macros</a>	Macros are a way to extend the Lisp language.
<a href="#">Customization</a>	Making variables and faces customizable.
<a href="#">Loading</a>	Reading files of Lisp code into Lisp.
<a href="#">Byte Compilation</a>	Compilation makes programs run faster.
<a href="#">Debugging</a>	Tools and tips for debugging Lisp programs.
<a href="#">Read and Print</a>	Converting Lisp objects to text and back.
<a href="#">Minibuffers</a>	Using the minibuffer to read input.
<a href="#">Command Loop</a>	How the editor command loop works, and how you can call its subroutines.
<a href="#">Keymaps</a>	Defining the bindings from keys to commands.
<a href="#">Modes</a>	Defining major and minor modes.
<a href="#">Documentation</a>	Writing and using documentation strings.
<a href="#">Files</a>	Accessing files.
<a href="#">Backups and Auto-Saving</a>	Controlling how backups and auto-save files are made.
<a href="#">Buffers</a>	Creating and using buffer objects.
<a href="#">Windows</a>	Manipulating windows and displaying buffers.
<a href="#">Frames</a>	Making multiple system-level windows.
<a href="#">Positions</a>	Buffer positions and motion functions.

<a href="#">Markers</a>	Markers represent positions and update automatically when the text is changed.
<a href="#">Text</a>	Examining and changing text in buffers.
<a href="#">Non-ASCII Characters</a>	Non-ASCII text in buffers and strings.
<a href="#">Searching and Matching</a>	Searching buffers for strings or regexps.
<a href="#">Syntax Tables</a>	The syntax table controls word and list parsing.
<a href="#">Abbrevs</a>	How Abbrev mode works, and its data structures.
<a href="#">Processes</a>	Running and communicating with subprocesses.
<a href="#">Display</a>	Features for controlling the screen display.
<a href="#">System Interface</a>	Getting the user id, system type, environment variables, and other such things.
<a href="#">Packaging</a>	Preparing Lisp code for distribution.
<b>Appendices</b>	
<a href="#">Antinews</a>	Info for users downgrading to Emacs 24.
<a href="#">GNU Free Documentation License</a>	The license for this documentation.
<a href="#">GPL</a>	Conditions for copying and changing GNU Emacs.
<a href="#">Tips</a>	Advice and coding conventions for Emacs Lisp.
<a href="#">GNU Emacs Internals</a>	Building and dumping Emacs; internal data structures.
<a href="#">Standard Errors</a>	List of some standard error symbols.
<a href="#">Standard Keymaps</a>	List of some standard keymaps.
<a href="#">Standard Hooks</a>	List of some standard hook variables.
<a href="#">Index</a>	Index including concepts, functions, variables, and other terms.

## Detailed Node Listing

**Here are other nodes that are subnodes of those already listed, mentioned here so you can get to them in one step:**

### Introduction

<a href="#">Caveats</a>	Flaws and a request for help.
<a href="#">Lisp History</a>	Emacs Lisp is descended from Maclisp.
<a href="#">Conventions</a>	How the manual is formatted.
<a href="#">Version Info</a>	Which Emacs version is running?
<a href="#">Acknowledgments</a>	The authors, editors, and sponsors of this manual.

### Conventions

<a href="#">Some Terms</a>	Explanation of terms we use in this manual.
<a href="#">nil and t</a>	How the symbols <code>nil</code> and <code>t</code> are used.
<a href="#">Evaluation Notation</a>	The format we use for examples of evaluation.

<a href="#">Printing Notation</a>	The format we use when examples print text.
<a href="#">Error Messages</a>	The format we use for examples of errors.
<a href="#">Buffer Text Notation</a>	The format we use for buffer contents in examples.
<a href="#">Format of Descriptions</a>	Notation for describing functions, variables, etc.

## Format of Descriptions

<a href="#">A Sample Function Description</a>	A description of an imaginary function, <code>foo</code> .
<a href="#">A Sample Variable Description</a>	A description of an imaginary variable, <code>electric-future-map</code> .

## Lisp Data Types

<a href="#">Printed Representation</a>	How Lisp objects are represented as text.
<a href="#">Comments</a>	Comments and their formatting conventions.
<a href="#">Programming Types</a>	Types found in all Lisp systems.
<a href="#">Editing Types</a>	Types specific to Emacs.
<a href="#">Circular Objects</a>	Read syntax for circular structure.
<a href="#">Type Predicates</a>	Tests related to types.
<a href="#">Equality Predicates</a>	Tests of equality between any two objects.

## Programming Types

<a href="#">Integer Type</a>	Numbers without fractional parts.
<a href="#">Floating-Point Type</a>	Numbers with fractional parts and with a large range.
<a href="#">Character Type</a>	The representation of letters, numbers and control characters.
<a href="#">Symbol Type</a>	A multi-use object that refers to a function, variable, or property list, and has a unique identity.
<a href="#">Sequence Type</a>	Both lists and arrays are classified as sequences.
<a href="#">Cons Cell Type</a>	Cons cells, and lists (which are made from cons cells).
<a href="#">Array Type</a>	Arrays include strings and vectors.
<a href="#">String Type</a>	An (efficient) array of characters.
<a href="#">Vector Type</a>	One-dimensional arrays.
<a href="#">Char-Table Type</a>	One-dimensional sparse arrays indexed by characters.
<a href="#">Bool-Vector Type</a>	One-dimensional arrays of <code>t</code> or <code>nil</code> .
<a href="#">Hash Table Type</a>	Super-fast lookup tables.
<a href="#">Function Type</a>	A piece of executable code you can call from elsewhere.
<a href="#">Macro Type</a>	A method of expanding an expression into another expression, more fundamental but less pretty.
<a href="#">Primitive Function Type</a>	A function written in C, callable from Lisp.
<a href="#">Byte-Code Type</a>	A function written in Lisp, then compiled.
<a href="#">Autoload Type</a>	A type used for automatically loading seldom-used functions.
<a href="#">Finalizer Type</a>	Runs code when no longer reachable.

## Character Type

<a href="#">Basic Char Syntax</a>	Syntax for regular characters.
<a href="#">General Escape Syntax</a>	How to specify characters by their codes.
<a href="#">Ctl-Char Syntax</a>	Syntax for control characters.
<a href="#">Meta-Char Syntax</a>	Syntax for meta-characters.
<a href="#">Other Char Bits</a>	Syntax for hyper-, super-, and alt-characters.

## Cons Cell and List Types

<a href="#">Box Diagrams</a>	Drawing pictures of lists.
<a href="#">Dotted Pair Notation</a>	A general syntax for cons cells.
<a href="#">Association List Type</a>	A specially constructed list.

## String Type

<a href="#">Syntax for Strings</a>	How to specify Lisp strings.
<a href="#">Non-ASCII in Strings</a>	International characters in strings.
<a href="#">Nonprinting Characters</a>	Literal unprintable characters in strings.
<a href="#">Text Props and Strings</a>	Strings with text properties.

## Editing Types

<a href="#">Buffer Type</a>	The basic object of editing.
<a href="#">Marker Type</a>	A position in a buffer.
<a href="#">Window Type</a>	Buffers are displayed in windows.
<a href="#">Frame Type</a>	Windows subdivide frames.
<a href="#">Terminal Type</a>	A terminal device displays frames.
<a href="#">Window Configuration Type</a>	Recording the way a frame is subdivided.
<a href="#">Frame Configuration Type</a>	Recording the status of all frames.
<a href="#">Process Type</a>	A subprocess of Emacs running on the underlying OS.
<a href="#">Stream Type</a>	Receive or send characters.
<a href="#">Keymap Type</a>	What function a keystroke invokes.
<a href="#">Overlay Type</a>	How an overlay is represented.
<a href="#">Font Type</a>	Fonts for displaying text.

## Numbers

<a href="#">Integer Basics</a>	Representation and range of integers.
<a href="#">Float Basics</a>	Representation and range of floating point.
<a href="#">Predicates on Numbers</a>	Testing for numbers.
<a href="#">Comparison of Numbers</a>	Equality and inequality predicates.
<a href="#">Numeric Conversions</a>	Converting float to integer and vice versa.

<a href="#">Arithmetic Operations</a>	How to add, subtract, multiply and divide.
<a href="#">Rounding Operations</a>	Explicitly rounding floating-point numbers.
<a href="#">Bitwise Operations</a>	Logical and, or, not, shifting.
<a href="#">Math Functions</a>	Trig, exponential and logarithmic functions.
<a href="#">Random Numbers</a>	Obtaining random integers, predictable or not.

## Strings and Characters

<a href="#">String Basics</a>	Basic properties of strings and characters.
<a href="#">Predicates for Strings</a>	Testing whether an object is a string or char.
<a href="#">Creating Strings</a>	Functions to allocate new strings.
<a href="#">Modifying Strings</a>	Altering the contents of an existing string.
<a href="#">Text Comparison</a>	Comparing characters or strings.
<a href="#">String Conversion</a>	Converting to and from characters and strings.
<a href="#">Formatting Strings</a>	format: Emacs's analogue of printf.
<a href="#">Case Conversion</a>	Case conversion functions.
<a href="#">Case Tables</a>	Customizing case conversion.

## Lists

<a href="#">Cons Cells</a>	How lists are made out of cons cells.
<a href="#">List-related Predicates</a>	Is this object a list? Comparing two lists.
<a href="#">List Elements</a>	Extracting the pieces of a list.
<a href="#">Building Lists</a>	Creating list structure.
<a href="#">List Variables</a>	Modifying lists stored in variables.
<a href="#">Modifying Lists</a>	Storing new pieces into an existing list.
<a href="#">Sets And Lists</a>	A list can represent a finite mathematical set.
<a href="#">Association Lists</a>	A list can represent a finite relation or mapping.
<a href="#">Property Lists</a>	A list of paired elements.

## Modifying Existing List Structure

<a href="#">Setcar</a>	Replacing an element in a list.
<a href="#">Setcdr</a>	Replacing part of the list backbone. This can be used to remove or add elements.
<a href="#">Rearrangement</a>	Reordering the elements in a list; combining lists.

## Property Lists

<a href="#">Plists and Alists</a>	Comparison of the advantages of property lists and association lists.
<a href="#">Plist Access</a>	Accessing property lists stored elsewhere.

## Sequences, Arrays, and Vectors

<a href="#">Sequence Functions</a>	Functions that accept any kind of sequence.
<a href="#">Arrays</a>	Characteristics of arrays in Emacs Lisp.
<a href="#">Array Functions</a>	Functions specifically for arrays.
<a href="#">Vectors</a>	Special characteristics of Emacs Lisp vectors.

## [Vector Functions](#)

Functions specifically for vectors.

## [Char-Tables](#)

How to work with char-tables.

## [Bool-Vectors](#)

How to work with bool-vectors.

## [Rings](#)

Managing a fixed-size ring of objects.

## Hash Tables

### [Creating Hash](#)

Functions to create hash tables.

### [Hash Access](#)

Reading and writing the hash table contents.

### [Defining Hash](#)

Defining new comparison methods.

### [Other Hash](#)

Miscellaneous.

## Symbols

### [Symbol Components](#)

Symbols have names, values, function definitions and property lists.

### [Definitions](#)

A definition says how a symbol will be used.

### [Creating Symbols](#)

How symbols are kept unique.

### [Symbol Properties](#)

Each symbol has a property list for recording miscellaneous information.

## Symbol Properties

### [Symbol Plists](#)

Accessing symbol properties.

### [Standard Properties](#)

Standard meanings of symbol properties.

## Evaluation

### [Intro Eval](#)

Evaluation in the scheme of things.

### [Forms](#)

How various sorts of objects are evaluated.

### [Quoting](#)

Avoiding evaluation (to put constants in the program).

### [Backquote](#)

Easier construction of list structure.

### [Eval](#)

How to invoke the Lisp interpreter explicitly.

## Kinds of Forms

### [Self-Evaluating Forms](#)

Forms that evaluate to themselves.

### [Symbol Forms](#)

Symbols evaluate as variables.

### [Classifying Lists](#)

How to distinguish various sorts of list forms.

### [Function Indirection](#)

When a symbol appears as the car of a list, we find the real function via the symbol.

### [Function Forms](#)

Forms that call functions.

### [Macro Forms](#)

Forms that call macros.

### [Special Forms](#)

Special forms are idiosyncratic primitives, most of them extremely important.

### [Autoloading](#)

Functions set up to load files containing their real definitions.

## Control Structures

### [Sequencing](#)

Evaluation in textual order.

### [Conditionals](#)

if, cond, when, unless.

### [Combining Conditions](#)

and, or, not.

### [Iteration](#)

while loops.

[Generators](#)

Generic sequences and coroutines.

[Nonlocal Exits](#)

Jumping out of a sequence.

**Conditionals**[Pattern matching case statement](#)

How to use `pcase`.

**Nonlocal Exits**[Catch and Throw](#)

Nonlocal exits for the program's own purposes.

[Examples of Catch](#)

Showing how such nonlocal exits can be written.

[Errors](#)

How errors are signaled and handled.

[Cleanups](#)

Arranging to run a cleanup form if an error happens.

**Errors**[Signaling Errors](#)

How to report an error.

[Processing of Errors](#)

What Emacs does when you report an error.

[Handling Errors](#)

How you can trap errors and continue execution.

[Error Symbols](#)

How errors are classified for trapping them.

**Variables**[Global Variables](#)

Variable values that exist permanently, everywhere.

[Constant Variables](#)

Variables that never change.

[Local Variables](#)

Variable values that exist only temporarily.

[Void Variables](#)

Symbols that lack values.

[Defining Variables](#)

A definition says a symbol is used as a variable.

[Tips for Defining](#)

Things you should think about when you define a variable.

[Accessing Variables](#)

Examining values of variables whose names are known only at run time.

[Setting Variables](#)

Storing new values in variables.

[Variable Scoping](#)

How Lisp chooses among local and global values.

[Buffer-Local Variables](#)

Variable values in effect only in one buffer.

[File Local Variables](#)

Handling local variable lists in files.

[Directory Local Variables](#)

Local variables common to all files in a directory.

[Variable Aliases](#)

Variables that are aliases for other variables.

[Variables with Restricted Values](#)

Non-constant variables whose value can *not* be an arbitrary Lisp object.

[Generalized Variables](#)

Extending the concept of variables.

**Scoping Rules for Variable Bindings**[Dynamic Binding](#)

The default for binding local variables in Emacs.

[Dynamic Binding Tips](#)

Avoiding problems with dynamic binding.

[Lexical Binding](#)

A different type of local variable binding.

[Using Lexical Binding](#)

How to enable lexical binding.



## Buffer-Local Variables

<a href="#">Intro to Buffer-Local</a>	Introduction and concepts.
<a href="#">Creating Buffer-Local</a>	Creating and destroying buffer-local bindings.
<a href="#">Default Value</a>	The default value is seen in buffers that don't have their own buffer-local values.

## Generalized Variables

<a href="#">Setting Generalized Variables</a>	The <code>setf</code> macro.
<a href="#">Adding Generalized Variables</a>	Defining new <code>setf</code> forms.

## Functions

<a href="#">What Is a Function</a>	Lisp functions vs. primitives; terminology.
<a href="#">Lambda Expressions</a>	How functions are expressed as Lisp objects.
<a href="#">Function Names</a>	A symbol can serve as the name of a function.
<a href="#">Defining Functions</a>	Lisp expressions for defining functions.
<a href="#">Calling Functions</a>	How to use an existing function.
<a href="#">Mapping Functions</a>	Applying a function to each element of a list, etc.
<a href="#">Anonymous Functions</a>	Lambda expressions are functions with no names.
<a href="#">Generic Functions</a>	Polymorphism, Emacs-style.
<a href="#">Function Cells</a>	Accessing or setting the function definition of a symbol.
<a href="#">Closures</a>	Functions that enclose a lexical environment.
<a href="#">Advising Functions</a>	Adding to the definition of a function.
<a href="#">Obsolete Functions</a>	Declaring functions obsolete.
<a href="#">Inline Functions</a>	Defining functions that the compiler will expand inline.
<a href="#">Declare Form</a>	Adding additional information about a function.
<a href="#">Declaring Functions</a>	Telling the compiler that a function is defined.
<a href="#">Function Safety</a>	Determining whether a function is safe to call.
<a href="#">Related Topics</a>	Cross-references to specific Lisp primitives that have a special bearing on how functions work.

## Lambda Expressions

<a href="#">Lambda Components</a>	The parts of a lambda expression.
<a href="#">Simple Lambda</a>	A simple example.
<a href="#">Argument List</a>	Details and special features of argument lists.
<a href="#">Function Documentation</a>	How to put documentation in a function.

## Advising Emacs Lisp Functions

<a href="#">Core Advising Primitives</a>	Primitives to manipulate advice.
<a href="#">Advising Named Functions</a>	Advising named functions.



[Advice combinators](#)

Ways to compose advice.

[Porting old advice](#)

Adapting code using the old defadvice.

## Macros

[Simple Macro](#)

A basic example.

[Expansion](#)

How, when and why macros are expanded.

[Compiling Macros](#)

How macros are expanded by the compiler.

[Defining Macros](#)

How to write a macro definition.

[Problems with  
Macros](#)

Don't evaluate the macro arguments too many times. Don't hide the user's variables.

[Indenting Macros](#)

Specifying how to indent macro calls.

## Common Problems Using Macros

[Wrong Time](#)

Do the work in the expansion, not in the macro.

[Argument Evaluation](#)

The expansion should evaluate each macro arg once.

[Surprising Local Vars](#)

Local variable bindings in the expansion require special care.

[Eval During  
Expansion](#)

Don't evaluate them; put them in the expansion.

[Repeated Expansion](#)

Avoid depending on how many times expansion is done.

## Customization Settings

[Common Keywords](#)

Common keyword arguments for all kinds of customization declarations.

[Group Definitions](#)

Writing customization group definitions.

[Variable Definitions](#)

Declaring user options.

[Customization Types](#)

Specifying the type of a user option.

[Applying  
Customizations](#)

Functions to apply customization settings.

[Custom Themes](#)

Writing Custom themes.

## Customization Types

[Simple Types](#)

Simple customization types: sexp, integer, etc.

[Composite Types](#)

Build new types from other types or data.

[Splicing into Lists](#)

Splice elements into list with `:inline`.

[Type Keywords](#)

Keyword-argument pairs in a customization type.

[Defining New Types](#)

Give your type a name.

## Loading

[How Programs Do  
Loading](#)

The `load` function and others.

[Load Suffixes](#)

Details about the suffixes that `load` tries.

[Library Search](#)

Finding a library to load.

[Loading Non-ASCII](#)

Non-ASCII characters in Emacs Lisp files.

[Autoload](#)

Setting up a function to autoload.

[Repeated Loading](#)

Precautions about loading a file twice.

[Named Features](#)

Loading a library if it isn't already loaded.

<a href="#">Where Defined</a>	Finding which file defined a certain symbol.
<a href="#">Unloading</a>	How to unload a library that was loaded.
<a href="#">Hooks for Loading</a>	Providing code to be run when particular libraries are loaded.
<a href="#">Dynamic Modules</a>	Modules provide additional Lisp primitives.

## Byte Compilation

<a href="#">Speed of Byte-Code</a>	An example of speedup from byte compilation.
<a href="#">Compilation Functions</a>	Byte compilation functions.
<a href="#">Docs and Compilation</a>	Dynamic loading of documentation strings.
<a href="#">Dynamic Loading</a>	Dynamic loading of individual functions.
<a href="#">Eval During Compile</a>	Code to be evaluated when you compile.
<a href="#">Compiler Errors</a>	Handling compiler error messages.
<a href="#">Byte-Code Objects</a>	The data type used for byte-compiled functions.
<a href="#">Disassembly</a>	Disassembling byte-code; how to read byte-code.

## Debugging Lisp Programs

<a href="#">Debugger</a>	A debugger for the Emacs Lisp evaluator.
<a href="#">Edebug</a>	A source-level Emacs Lisp debugger.
<a href="#">Syntax Errors</a>	How to find syntax errors.
<a href="#">Test Coverage</a>	Ensuring you have tested all branches in your code.
<a href="#">Profiling</a>	Measuring the resources that your code uses.

## The Lisp Debugger

<a href="#">Error Debugging</a>	Entering the debugger when an error happens.
<a href="#">Infinite Loops</a>	Stopping and debugging a program that doesn't exit.
<a href="#">Function Debugging</a>	Entering it when a certain function is called.
<a href="#">Explicit Debug</a>	Entering it at a certain point in the program.
<a href="#">Using Debugger</a>	What the debugger does; what you see while in it.
<a href="#">Debugger Commands</a>	Commands used while in the debugger.
<a href="#">Invoking the Debugger</a>	How to call the function debug.
<a href="#">Internals of Debugger</a>	Subroutines of the debugger, and global variables.

## Edebug

<a href="#">Using Edebug</a>	Introduction to use of Edebug.
<a href="#">Instrumenting</a>	You must instrument your code in order to debug it with Edebug.
<a href="#">Edebug Execution Modes</a>	Execution modes, stopping more or less often.
<a href="#">Jumping</a>	Commands to jump to a specified place.
<a href="#">Edebug Misc</a>	Miscellaneous commands.
<a href="#">Breaks</a>	Setting breakpoints to make the program stop.
<a href="#">Trapping Errors</a>	Trapping errors with Edebug.
<a href="#">Edebug Views</a>	Views inside and outside of Edebug.

<a href="#">Edebug Eval</a>	Evaluating expressions within Edebug.
<a href="#">Eval List</a>	Expressions whose values are displayed each time you enter Edebug.
<a href="#">Printing in Edebug</a>	Customization of printing.
<a href="#">Trace Buffer</a>	How to produce trace output in a buffer.
<a href="#">Coverage Testing</a>	How to test evaluation coverage.
<a href="#">The Outside Context</a>	Data that Edebug saves and restores.
<a href="#">Edebug and Macros</a>	Specifying how to handle macro calls.
<a href="#">Edebug Options</a>	Option variables for customizing Edebug.

## Breaks

<a href="#">Breakpoints</a>	Breakpoints at stop points.
<a href="#">Global Break Condition</a>	Breaking on an event.
<a href="#">Source Breakpoints</a>	Embedding breakpoints in source code.

## The Outside Context

<a href="#">Checking Whether to Stop</a>	When Edebug decides what to do.
<a href="#">Edebug Display Update</a>	When Edebug updates the display.
<a href="#">Edebug Recursive Edit</a>	When Edebug stops execution.

## Edebug and Macros

<a href="#">Instrumenting Macro Calls</a>	The basic problem.
<a href="#">Specification List</a>	How to specify complex patterns of evaluation.
<a href="#">Backtracking</a>	What Edebug does when matching fails.
<a href="#">Specification Examples</a>	To help understand specifications.

## Debugging Invalid Lisp Syntax

<a href="#">Excess Open</a>	How to find a spurious open paren or missing close.
<a href="#">Excess Close</a>	How to find a spurious close paren or missing open.

## Reading and Printing Lisp Objects

<a href="#">Streams Intro</a>	Overview of streams, reading and printing.
<a href="#">Input Streams</a>	Various data types that can be used as input streams.
<a href="#">Input Functions</a>	Functions to read Lisp objects from text.
<a href="#">Output Streams</a>	Various data types that can be used as output streams.
<a href="#">Output Functions</a>	Functions to print Lisp objects as text.
<a href="#">Output Variables</a>	Variables that control what the printing functions do.

## Minibuffers

<a href="#">Intro to Minibuffers</a>	Basic information about minibuffers.
<a href="#">Text from Minibuffer</a>	How to read a straight text string.
<a href="#">Object from</a>	How to read a Lisp object or expression.

[Minibuffer](#)[Minibuffer History](#)

Recording previous minibuffer inputs so the user can reuse them.

[Initial Input](#)

Specifying initial contents for the minibuffer.

[Completion](#)

How to invoke and customize completion.

[Yes-or-No Queries](#)

Asking a question with a simple answer.

[Multiple Queries](#)

Asking a series of similar questions.

[Reading a Password](#)

Reading a password from the terminal.

[Minibuffer](#)[Commands](#)

Commands used as key bindings in minibuffers.

[Minibuffer Windows](#)

Operating on the special minibuffer windows.

[Minibuffer Contents](#)

How such commands access the minibuffer text.

[Recursive Mini](#)

Whether recursive entry to minibuffer is allowed.

[Minibuffer Misc](#)

Various customization hooks and variables.

**Completion**[Basic Completion](#)

Low-level functions for completing strings.

[Minibuffer](#)[Completion](#)

Invoking the minibuffer with completion.

[Completion](#)[Commands](#)

Minibuffer commands that do completion.

[High-Level](#)[Completion](#)

Convenient special cases of completion (reading buffer names, variable names, etc.).

[Reading File Names](#)

Using completion to read file names and shell commands.

[Completion Variables](#)

Variables controlling completion behavior.

[Programmed](#)[Completion](#)

Writing your own completion function.

[Completion in](#)[Buffers](#)

Completing text in ordinary buffers.

**Command Loop**[Command Overview](#)

How the command loop reads commands.

[Defining Commands](#)

Specifying how a function should read arguments.

[Interactive Call](#)

Calling a command, so that it will read arguments.

[Distinguish](#)[Interactive](#)

Making a command distinguish interactive calls.

[Command Loop Info](#)

Variables set by the command loop for you to examine.

[Adjusting Point](#)

Adjustment of point after a command.

[Input Events](#)

What input looks like when you read it.

[Reading Input](#)

How to read input events from the keyboard or mouse.

[Special Events](#)

Events processed immediately and individually.

[Waiting](#)

Waiting for user input or elapsed time.

[Quitting](#)

How c-g works. How to catch or defer quitting.

[Prefix Command](#)[Arguments](#)

How the commands to set prefix args work.

<a href="#">Recursive Editing</a>	Entering a recursive edit, and why you usually shouldn't.
<a href="#">Disabling Commands</a>	How the command loop handles disabled commands.
<a href="#">Command History</a>	How the command history is set up, and how accessed.
<a href="#">Keyboard Macros</a>	How keyboard macros are implemented.
<b>Defining Commands</b>	
<a href="#">Using Interactive</a>	General rules for <code>interactive</code> .
<a href="#">Interactive Codes</a>	The standard letter-codes for reading arguments in various ways.
<a href="#">Interactive Examples</a>	Examples of how to read interactive arguments.
<a href="#">Generic Commands</a>	Select among command alternatives.
<b>Input Events</b>	
<a href="#">Keyboard Events</a>	Ordinary characters -- keys with symbols on them.
<a href="#">Function Keys</a>	Function keys -- keys with names, not symbols.
<a href="#">Mouse Events</a>	Overview of mouse events.
<a href="#">Click Events</a>	Pushing and releasing a mouse button.
<a href="#">Drag Events</a>	Moving the mouse before releasing the button.
<a href="#">Button-Down Events</a>	A button was pushed and not yet released.
<a href="#">Repeat Events</a>	Double and triple click (or drag, or down).
<a href="#">Motion Events</a>	Just moving the mouse, not pushing a button.
<a href="#">Focus Events</a>	Moving the mouse between frames.
<a href="#">Misc Events</a>	Other events the system can generate.
<a href="#">Event Examples</a>	Examples of the lists for mouse events.
<a href="#">Classifying Events</a>	Finding the modifier keys in an event symbol. Event types.
<a href="#">Accessing Mouse</a>	Functions to extract info from mouse events.
<a href="#">Accessing Scroll</a>	Functions to get info from scroll bar events.
<a href="#">Strings of Events</a>	Special considerations for putting keyboard character events in a string.
<b>Reading Input</b>	
<a href="#">Key Sequence Input</a>	How to read one key sequence.
<a href="#">Reading One Event</a>	How to read just one event.
<a href="#">Event Mod</a>	How Emacs modifies events as they are read.
<a href="#">Invoking the Input Method</a>	How reading an event uses the input method.
<a href="#">Quoted Character Input</a>	Asking the user to specify a character.
<a href="#">Event Input Misc</a>	How to reread or throw away input events.
<b>Keymaps</b>	
<a href="#">Key Sequences</a>	Key sequences as Lisp objects.
<a href="#">Keymap Basics</a>	Basic concepts of keymaps.
<a href="#">Format of Keymaps</a>	What a keymap looks like as a Lisp object.
<a href="#">Creating Keymaps</a>	Functions to create and copy keymaps.
<a href="#">Inheritance and</a>	How one keymap can inherit the bindings of another keymap.

## [Keymaps](#)

### [Prefix Keys](#)

Defining a key with a keymap as its definition.

### [Active Keymaps](#)

How Emacs searches the active keymaps for a key binding.

### [Searching Keymaps](#)

A pseudo-Lisp summary of searching active maps.

### [Controlling Active Maps](#)

Each buffer has a local keymap to override the standard (global) bindings. A minor mode can also override them.

### [Key Lookup](#)

Finding a key's binding in one keymap.

### [Functions for Key Lookup](#)

How to request key lookup.

### [Changing Key Bindings](#)

Redefining a key in a keymap.

### [Remapping Commands](#)

A keymap can translate one command to another.

### [Translation Keymaps](#)

Keymaps for translating sequences of events.

### [Key Binding Commands](#)

Interactive interfaces for redefining keys.

### [Scanning Keymaps](#)

Looking through all keymaps, for printing help.

### [Menu Keymaps](#)

Defining a menu as a keymap.

## **Menu Keymaps**

### [Defining Menus](#)

How to make a keymap that defines a menu.

### [Mouse Menus](#)

How users actuate the menu with the mouse.

### [Keyboard Menus](#)

How users actuate the menu with the keyboard.

### [Menu Example](#)

Making a simple menu.

### [Menu Bar](#)

How to customize the menu bar.

### [Tool Bar](#)

A tool bar is a row of images.

### [Modifying Menus](#)

How to add new items to a menu.

### [Easy Menu](#)

A convenience macro for defining menus.

## **Defining Menus**

### [Simple Menu Items](#)

A simple kind of menu key binding.

### [Extended Menu Items](#)

More complex menu item definitions.

### [Menu Separators](#)

Drawing a horizontal line through a menu.

### [Alias Menu Items](#)

Using command aliases in menu items.

## **Major and Minor Modes**

### [Hooks](#)

How to use hooks; how to write code that provides hooks.

### [Major Modes](#)

Defining major modes.

### [Minor Modes](#)

Defining minor modes.

### [Mode Line Format](#)

Customizing the text that appears in the mode line.

### [Imenu](#)

Providing a menu of definitions made in a buffer.

### [Font Lock Mode](#)

How modes can highlight text according to syntax.

### [Auto-Indentation](#)

How to teach Emacs to indent for a major mode.

[Desktop Save Mode](#)

How modes can have buffer state saved between Emacs sessions.

**Hooks**[Running Hooks](#)

How to run a hook.

[Setting Hooks](#)

How to put functions on a hook, or remove them.

**Major Modes**[Major Mode Conventions](#)

Coding conventions for keymaps, etc.

[Auto Major Mode](#)

How Emacs chooses the major mode automatically.

[Mode Help](#)

Finding out how to use a mode.

[Derived Modes](#)

Defining a new major mode based on another major mode.

[Basic Major Modes](#)

Modes that other modes are often derived from.

[Mode Hooks](#)

Hooks run at the end of major mode functions.

[Tabulated List Mode](#)

Parent mode for buffers containing tabulated data.

[Generic Modes](#)

Defining a simple major mode that supports comment syntax and Font Lock mode.

[Example Major Modes](#)

Text mode and Lisp modes.

**Minor Modes**[Minor Mode Conventions](#)

Tips for writing a minor mode.

[Keymaps and Minor Modes](#)

How a minor mode can have its own keymap.

[Defining Minor Modes](#)

A convenient facility for defining minor modes.

**Mode Line Format**[Mode Line Basics](#)

Basic ideas of mode line control.

[Mode Line Data](#)

The data structure that controls the mode line.

[Mode Line Top](#)

The top level variable, mode-line-format.

[Mode Line Variables](#)

Variables used in that data structure.

[%-Constructs](#)

Putting information into a mode line.

[Properties in Mode](#)

Using text properties in the mode line.

[Header Lines](#)

Like a mode line, but at the top.

[Emulating Mode Line](#)

Formatting text as the mode line would.

**Font Lock Mode**[Font Lock Basics](#)

Overview of customizing Font Lock.

[Search-based Fontification](#)

Fontification based on regexps.

[Customizing Keywords](#)

Customizing search-based fontification.

[Other Font Lock Variables](#)

Additional customization facilities.



<a href="#">Levels of Font Lock</a>	Each mode can define alternative levels so that the user can select more or less.
<a href="#">Precalculated Fontification</a>	How Lisp programs that produce the buffer contents can also specify how to fontify it.
<a href="#">Faces for Font Lock</a>	Special faces specifically for Font Lock.
<a href="#">Syntactic Font Lock</a>	Fontification based on syntax tables.
<a href="#">Multiline Font Lock</a>	How to coerce Font Lock into properly highlighting multiline constructs.

### **Multiline Font Lock Constructs**

<a href="#">Font Lock Multiline</a>	Marking multiline chunks with a text property.
<a href="#">Region to Refontify</a>	Controlling which region gets refontified after a buffer change.

### **Automatic Indentation of code**

<a href="#">SMIE</a>	A simple minded indentation engine.
----------------------	-------------------------------------

### **Simple Minded Indentation Engine**

<a href="#">SMIE setup</a>	SMIE setup and features.
<a href="#">Operator Precedence Grammars</a>	A very simple parsing technique.
<a href="#">SMIE Grammar</a>	Defining the grammar of a language.
<a href="#">SMIE Lexer</a>	Defining tokens.
<a href="#">SMIE Tricks</a>	Working around the parser's limitations.
<a href="#">SMIE Indentation</a>	Specifying indentation rules.
<a href="#">SMIE Indentation Helpers</a>	Helper functions for indentation rules.
<a href="#">SMIE Indentation Example</a>	Sample indentation rules.
<a href="#">SMIE Customization</a>	Customizing indentation.

### **Documentation**

<a href="#">Documentation Basics</a>	Where doc strings are defined and stored.
<a href="#">Accessing Documentation</a>	How Lisp programs can access doc strings.
<a href="#">Keys in Documentation</a>	Substituting current key bindings.
<a href="#">Describing Characters</a>	Making printable descriptions of non-printing characters and key sequences.
<a href="#">Help Functions</a>	Subroutines used by Emacs help facilities.

### **Files**

<a href="#">Visiting Files</a>	Reading files into Emacs buffers for editing.
<a href="#">Saving Buffers</a>	Writing changed buffers back into files.
<a href="#">Reading from Files</a>	Reading files into buffers without visiting.
<a href="#">Writing to Files</a>	Writing new files from parts of buffers.
<a href="#">File Locks</a>	Locking and unlocking files, to prevent simultaneous editing by two people.
<a href="#">Information about Files</a>	Testing existence, accessibility, size of files.

<a href="#">Changing Files</a>	Renaming files, changing permissions, etc.
<a href="#">File Names</a>	Decomposing and expanding file names.
<a href="#">Contents of Directories</a>	Getting a list of the files in a directory.
<a href="#">Create/Delete Dirs</a>	Creating and Deleting Directories.
<a href="#">Magic File Names</a>	Special handling for certain file names.
<a href="#">Format Conversion</a>	Conversion to and from various file formats.
<b>Visiting Files</b>	
<a href="#">Visiting Functions</a>	The usual interface functions for visiting.
<a href="#">Subroutines of Visiting</a>	Lower-level subroutines that they use.
<b>Information about Files</b>	
<a href="#">Testing Accessibility</a>	Is a given file readable? Writable?
<a href="#">Kinds of Files</a>	Is it a directory? A symbolic link?
<a href="#">Truenames</a>	Eliminating symbolic links from a file name.
<a href="#">File Attributes</a>	File sizes, modification times, etc.
<a href="#">Extended Attributes</a>	Extended file attributes for access control.
<a href="#">Locating Files</a>	How to find a file in standard places.
<b>File Names</b>	
<a href="#">File Name Components</a>	The directory part of a file name, and the rest.
<a href="#">Relative File Names</a>	Some file names are relative to a current directory.
<a href="#">Directory Names</a>	A directory's name as a directory is different from its name as a file.
<a href="#">File Name Expansion</a>	Converting relative file names to absolute ones.
<a href="#">Unique File Names</a>	Generating names for temporary files.
<a href="#">File Name Completion</a>	Finding the completions for a given file name.
<a href="#">Standard File Names</a>	If your package uses a fixed file name, how to handle various operating systems simply.
<b>File Format Conversion</b>	
<a href="#">Format Conversion Overview</a>	<code>insert-file-contents</code> and <code>write-region</code> .
<a href="#">Format Conversion Round-Trip</a>	Using <code>format-alist</code> .
<a href="#">Format Conversion Piecemeal</a>	Specifying non-paired conversion.
<b>Backups and Auto-Saving</b>	
<a href="#">Backup Files</a>	How backup files are made; how their names are chosen.
<a href="#">Auto-Saving</a>	How auto-save files are made; how their names are chosen.
<a href="#">Reverting</a>	<code>revert-buffer</code> , and how to customize what it does.
<b>Backup Files</b>	

<a href="#">Making Backups</a>	How Emacs makes backup files, and when.
<a href="#">Rename or Copy</a>	Two alternatives: renaming the old file or copying it.
<a href="#">Numbered Backups</a>	Keeping multiple backups for each source file.
<a href="#">Backup Names</a>	How backup file names are computed; customization.
<b>Buffers</b>	
<a href="#">Buffer Basics</a>	What is a buffer?
<a href="#">Current Buffer</a>	Designating a buffer as current so that primitives will access its contents.
<a href="#">Buffer Names</a>	Accessing and changing buffer names.
<a href="#">Buffer File Name</a>	The buffer file name indicates which file is visited.
<a href="#">Buffer Modification</a>	A buffer is modified if it needs to be saved.
<a href="#">Modification Time</a>	Determining whether the visited file was changed behind Emacs's back.
<a href="#">Read Only Buffers</a>	Modifying text is not allowed in a read-only buffer.
<a href="#">Buffer List</a>	How to look at all the existing buffers.
<a href="#">Creating Buffers</a>	Functions that create buffers.
<a href="#">Killing Buffers</a>	Buffers exist until explicitly killed.
<a href="#">Indirect Buffers</a>	An indirect buffer shares text with some other buffer.
<a href="#">Swapping Text</a>	Swapping text between two buffers.
<a href="#">Buffer Gap</a>	The gap in the buffer.
<b>Windows</b>	
<a href="#">Basic Windows</a>	Basic information on using windows.
<a href="#">Windows and Frames</a>	Relating windows to the frame they appear on.
<a href="#">Window Sizes</a>	Accessing a window's size.
<a href="#">Resizing Windows</a>	Changing the sizes of windows.
<a href="#">Preserving Window Sizes</a>	Preserving the size of windows.
<a href="#">Splitting Windows</a>	Splitting one window into two windows.
<a href="#">Deleting Windows</a>	Deleting a window gives its space to other windows.
<a href="#">Recombining Windows</a>	Preserving the frame layout when splitting and deleting windows.
<a href="#">Selecting Windows</a>	The selected window is the one that you edit in.
<a href="#">Cyclic Window Ordering</a>	Moving around the existing windows.
<a href="#">Buffers and Windows</a>	Each window displays the contents of a buffer.
<a href="#">Switching Buffers</a>	Higher-level functions for switching to a buffer.
<a href="#">Choosing Window</a>	How to choose a window for displaying a buffer.
<a href="#">Display Action Functions</a>	Subroutines for <code>display-buffer</code> .
<a href="#">Choosing Window Options</a>	Extra options affecting how buffers are displayed.
<a href="#">Window History</a>	Each window remembers the buffers displayed in it.
<a href="#">Dedicated Windows</a>	How to avoid displaying another buffer in a specific window.

<a href="#">Quitting Windows</a>	How to restore the state prior to displaying a buffer.
<a href="#">Window Point</a>	Each window has its own location of point.
<a href="#">Window Start and End</a>	Buffer positions indicating which text is on-screen in a window.
<a href="#">Textual Scrolling</a>	Moving text up and down through the window.
<a href="#">Vertical Scrolling</a>	Moving the contents up and down on the window.
<a href="#">Horizontal Scrolling</a>	Moving the contents sideways on the window.
<a href="#">Coordinates and Windows</a>	Converting coordinates to windows.
<a href="#">Window Configurations</a>	Saving and restoring the state of the screen.
<a href="#">Window Parameters</a>	Associating additional information with windows.
<a href="#">Window Hooks</a>	Hooks for scrolling, window size changes, redisplay going past a certain point, or window configuration changes.
<b>Frames</b>	
<a href="#">Creating Frames</a>	Creating additional frames.
<a href="#">Multiple Terminals</a>	Displaying on several different devices.
<a href="#">Frame Geometry</a>	Geometric properties of frames.
<a href="#">Frame Parameters</a>	Controlling frame size, position, font, etc.
<a href="#">Terminal Parameters</a>	Parameters common for all frames on terminal.
<a href="#">Frame Titles</a>	Automatic updating of frame titles.
<a href="#">Deleting Frames</a>	Frames last until explicitly deleted.
<a href="#">Finding All Frames</a>	How to examine all existing frames.
<a href="#">Minibuffers and Frames</a>	How a frame finds the minibuffer to use.
<a href="#">Input Focus</a>	Specifying the selected frame.
<a href="#">Visibility of Frames</a>	Frames may be visible or invisible, or icons.
<a href="#">Raising and Lowering</a>	Raising a frame makes it hide other windows; lowering it makes the others hide it.
<a href="#">Frame Configurations</a>	Saving the state of all frames.
<a href="#">Mouse Tracking</a>	Getting events that say when the mouse moves.
<a href="#">Mouse Position</a>	Asking where the mouse is, or moving it.
<a href="#">Pop-Up Menus</a>	Displaying a menu for the user to select from.
<a href="#">Dialog Boxes</a>	Displaying a box to ask yes or no.
<a href="#">Pointer Shape</a>	Specifying the shape of the mouse pointer.
<a href="#">Window System Selections</a>	Transferring text to and from other X clients.
<a href="#">Drag and Drop</a>	Internals of Drag-and-Drop implementation.
<a href="#">Color Names</a>	Getting the definitions of color names.
<a href="#">Text Terminal Colors</a>	Defining colors for text terminals.
<a href="#">Resources</a>	Getting resource values from the server.
<a href="#">Display Feature</a>	Determining the features of a terminal.

## [Testing](#)

### **Frame Geometry**

#### [Frame Layout](#)

Basic layout of frames.

#### [Frame Font](#)

The default font of a frame and how to set it.

#### [Size and Position](#)

Changing the size and position of a frame.

#### [Implied Frame](#)

#### [Resizing](#)

Implied resizing of frames and how to prevent it.

### **Frame Parameters**

#### [Parameter Access](#)

How to change a frame's parameters.

#### [Initial Parameters](#)

Specifying frame parameters when you make a frame.

#### [Window Frame](#)

#### [Parameters](#)

List of frame parameters for window systems.

#### [Geometry](#)

Parsing geometry specifications.

### **Window Frame Parameters**

#### [Basic Parameters](#)

Parameters that are fundamental.

#### [Position Parameters](#)

The position of the frame on the screen.

#### [Size Parameters](#)

Frame's size.

#### [Layout Parameters](#)

Size of parts of the frame, and enabling or disabling some parts.

#### [Buffer Parameters](#)

Which buffers have been or should be shown.

#### [Management](#)

#### [Parameters](#)

Communicating with the window manager.

#### [Cursor Parameters](#)

Controlling the cursor appearance.

#### [Font and Color](#)

#### [Parameters](#)

Fonts and colors for the frame text.

### **Positions**

#### [Point](#)

The special position where editing takes place.

#### [Motion](#)

Changing point.

#### [Excursions](#)

Temporary motion and buffer changes.

#### [Narrowing](#)

Restricting editing to a portion of the buffer.

### **Motion**

#### [Character Motion](#)

Moving in terms of characters.

#### [Word Motion](#)

Moving in terms of words.

#### [Buffer End Motion](#)

Moving to the beginning or end of the buffer.

#### [Text Lines](#)

Moving in terms of lines of text.

#### [Screen Lines](#)

Moving in terms of lines as displayed.

#### [List Motion](#)

Moving by parsing lists and sexps.

#### [Skipping Characters](#)

Skipping characters belonging to a certain set.

### **Markers**

#### [Overview of Markers](#)

The components of a marker, and how it relocates.

#### [Predicates on](#)

#### [Markers](#)

Testing whether an object is a marker.

<a href="#">Creating Markers</a>	Making empty markers or markers at certain places.
<a href="#">Information from Markers</a>	Finding the marker's buffer or character position.
<a href="#">Marker Insertion Types</a>	Two ways a marker can relocate when you insert where it points.
<a href="#">Moving Markers</a>	Moving the marker to a new buffer or position.
<a href="#">The Mark</a>	How the mark is implemented with a marker.
<a href="#">The Region</a>	How to access the region.
<b>Text</b>	
<a href="#">Near Point</a>	Examining text in the vicinity of point.
<a href="#">Buffer Contents</a>	Examining text in a general fashion.
<a href="#">Comparing Text</a>	Comparing substrings of buffers.
<a href="#">Insertion</a>	Adding new text to a buffer.
<a href="#">Commands for Insertion</a>	User-level commands to insert text.
<a href="#">Deletion</a>	Removing text from a buffer.
<a href="#">User-Level Deletion</a>	User-level commands to delete text.
<a href="#">The Kill Ring</a>	Where removed text sometimes is saved for later use.
<a href="#">Undo</a>	Undoing changes to the text of a buffer.
<a href="#">Maintaining Undo</a>	How to enable and disable undo information. How to control how much information is kept.
<a href="#">Filling</a>	Functions for explicit filling.
<a href="#">Margins</a>	How to specify margins for filling commands.
<a href="#">Adaptive Fill</a>	Adaptive Fill mode chooses a fill prefix from context.
<a href="#">Auto Filling</a>	How auto-fill mode is implemented to break lines.
<a href="#">Sorting</a>	Functions for sorting parts of the buffer.
<a href="#">Columns</a>	Computing horizontal positions, and using them.
<a href="#">Indentation</a>	Functions to insert or adjust indentation.
<a href="#">Case Changes</a>	Case conversion of parts of the buffer.
<a href="#">Text Properties</a>	Assigning Lisp property lists to text characters.
<a href="#">Substitution</a>	Replacing a given character wherever it appears.
<a href="#">Registers</a>	How registers are implemented. Accessing the text or position stored in a register.
<a href="#">Transposition</a>	Swapping two portions of a buffer.
<a href="#">Decompression</a>	Dealing with compressed data.
<a href="#">Base 64</a>	Conversion to or from base 64 encoding.
<a href="#">Checksum/Hash</a>	Computing cryptographic hashes.
<a href="#">Parsing HTML/XML</a>	Parsing HTML and XML.
<a href="#">Atomic Changes</a>	Installing several buffer changes atomically.
<a href="#">Change Hooks</a>	Supplying functions to be run when text is changed.
<b>The Kill Ring</b>	

<a href="#">Kill Ring Concepts</a>	What text looks like in the kill ring.
<a href="#">Kill Functions</a>	Functions that kill text.
<a href="#">Yanking</a>	How yanking is done.
<a href="#">Yank Commands</a>	Commands that access the kill ring.
<a href="#">Low-Level Kill Ring</a>	Functions and variables for kill ring access.
<a href="#">Internals of Kill Ring</a>	Variables that hold kill ring data.
<b>Indentation</b>	
<a href="#">Primitive Indent</a>	Functions used to count and insert indentation.
<a href="#">Mode-Specific Indent</a>	Customize indentation for different modes.
<a href="#">Region Indent</a>	Indent all the lines in a region.
<a href="#">Relative Indent</a>	Indent the current line based on previous lines.
<a href="#">Indent Tabs</a>	Adjustable, typewriter-like tab stops.
<a href="#">Motion by Indent</a>	Move to first non-blank character.
<b>Text Properties</b>	
<a href="#">Examining Properties</a>	Looking at the properties of one character.
<a href="#">Changing Properties</a>	Setting the properties of a range of text.
<a href="#">Property Search</a>	Searching for where a property changes value.
<a href="#">Special Properties</a>	Particular properties with special meanings.
<a href="#">Format Properties</a>	Properties for representing formatting of text.
<a href="#">Sticky Properties</a>	How inserted text gets properties from neighboring text.
<a href="#">Lazy Properties</a>	Computing text properties in a lazy fashion only when text is examined.
<a href="#">Clickable Text</a>	Using text properties to make regions of text do something when you click on them.
<a href="#">Fields</a>	The <code>field</code> property defines fields within the buffer.
<a href="#">Not Intervals</a>	Why text properties do not use Lisp-visible text intervals.
<b>Parsing HTML and XML</b>	
<a href="#">Document Object Model</a>	Access, manipulate and search the <u>DOM</u> .
<b>Non-ASCII Characters</b>	
<a href="#">Text Representations</a>	How Emacs represents text.
<a href="#">Disabling Multibyte</a>	Controlling whether to use multibyte characters.
<a href="#">Converting Representations</a>	Converting unibyte to multibyte and vice versa.
<a href="#">Selecting a Representation</a>	Treating a byte sequence as unibyte or multi.
<a href="#">Character Codes</a>	How unibyte and multibyte relate to codes of individual characters.
<a href="#">Character Properties</a>	Character attributes that define their behavior and handling.
<a href="#">Character Sets</a>	The space of possible character codes is divided into various character sets.
<a href="#">Scanning Charsets</a>	Which character sets are used in a buffer?
<a href="#">Translation of Characters</a>	Translation tables are used for conversion.



## [Coding Systems](#)

Coding systems are conversions for saving files.

## [Input Methods](#)

Input methods allow users to enter various non-ASCII characters without special keyboards.

## [Locales](#)

Interacting with the POSIX locale.

## **Coding Systems**

### [Coding System](#)

#### [Basics](#)

Basic concepts.

### [Encoding and I/O](#)

How file I/O functions handle coding systems.

### [Lisp and Coding Systems](#)

Functions to operate on coding system names.

### [User-Chosen Coding Systems](#)

Asking the user to choose a coding system.

### [Default Coding Systems](#)

Controlling the default choices.

### [Specifying Coding Systems](#)

Requesting a particular coding system for a single file operation.

### [Explicit Encoding](#)

Encoding or decoding text without doing I/O.

### [Terminal I/O Encoding](#)

Use of encoding for terminal I/O.

## **Searching and Matching**

### [String Search](#)

Search for an exact match.

### [Searching and Case](#)

Case-independent or case-significant searching.

### [Regular Expressions](#)

Describing classes of strings.

### [Regex Search](#)

Searching for a match for a regexp.

### [POSIX Regexp](#)

Searching POSIX-style for the longest match.

### [Match Data](#)

Finding out which part of the text matched, after a string or regexp search.

### [Search and Replace](#)

Commands that loop, searching and replacing.

### [Standard Regexp](#)

Useful regexps for finding sentences, pages,...

## **Regular Expressions**

### [Syntax of Regexp](#)

Rules for writing regular expressions.

### [Regex Example](#)

Illustrates regular expression syntax.

### [Regex Functions](#)

Functions for operating on regular expressions.

## **Syntax of Regular Expressions**

### [Regex Special](#)

Special characters in regular expressions.

### [Char Classes](#)

Character classes used in regular expressions.

### [Regex Backslash](#)

Backslash-sequences in regular expressions.

## **The Match Data**

### [Replacing Match](#)

Replacing a substring that was matched.

### [Simple Match Data](#)

Accessing single items of match data, such as where a particular subexpression started.

### [Entire Match Data](#)

Accessing the entire match data at once, as a list.

[Saving Match Data](#)

Saving and restoring the match data.

**Syntax Tables**[Syntax Basics](#)

Basic concepts of syntax tables.

[Syntax Descriptors](#)

How characters are classified.

[Syntax Table Functions](#)

How to create, examine and alter syntax tables.

[Syntax Properties](#)

Overriding syntax with text properties.

[Motion and Syntax](#)

Moving over characters with certain syntaxes.

[Parsing Expressions](#)

Parsing balanced expressions using the syntax table.

[Syntax Table Internals](#)

How syntax table information is stored.

[Categories](#)

Another way of classifying character syntax.

**Syntax Descriptors**[Syntax Class Table](#)

Table of syntax classes.

[Syntax Flags](#)

Additional flags each character can have.

**Parsing Expressions**[Motion via Parsing](#)

Motion functions that work by parsing.

[Position Parse](#)

Determining the syntactic state of a position.

[Parser State](#)

How Emacs represents a syntactic state.

[Low-Level Parsing](#)

Parsing across a specified region.

[Control Parsing](#)

Parameters that affect parsing.

**Abbrevs and Abbrev Expansion**[Abbrev Tables](#)

Creating and working with abbrev tables.

[Defining Abbrevs](#)

Specifying abbreviations and their expansions.

[Abbrev Files](#)

Saving abbrevs in files.

[Abbrev Expansion](#)

Controlling expansion; expansion subroutines.

[Standard Abbrev Tables](#)

Abbrev tables used by various major modes.

[Abbrev Properties](#)

How to read and set abbrev properties. Which properties have which effect.

[Abbrev Table Properties](#)

How to read and set abbrev table properties. Which properties have which effect.

**Processes**[Subprocess Creation](#)

Functions that start subprocesses.

[Shell Arguments](#)

Quoting an argument to pass it to a shell.

[Synchronous Processes](#)

Details of using synchronous subprocesses.

[Asynchronous Processes](#)

Starting up an asynchronous subprocess.

[Deleting Processes](#)

Eliminating an asynchronous subprocess.

[Process Information](#)

Accessing run-status and other attributes.

[Input to Processes](#)

Sending input to an asynchronous subprocess.

<a href="#">Signals to Processes</a>	Stopping, continuing or interrupting an asynchronous subprocess.
<a href="#">Output from Processes</a>	Collecting output from an asynchronous subprocess.
<a href="#">Sentinels</a>	Sentinels run when process run-status changes.
<a href="#">Query Before Exit</a>	Whether to query if exiting will kill a process.
<a href="#">System Processes</a>	Accessing other processes running on your system.
<a href="#">Transaction Queues</a>	Transaction-based communication with subprocesses.
<a href="#">Network</a>	Opening network connections.
<a href="#">Network Servers</a>	Network servers let Emacs accept net connections.
<a href="#">Datagrams</a>	UDP network connections.
<a href="#">Low-Level Network</a>	Lower-level but more general function to create connections and servers.
<a href="#">Misc Network</a>	Additional relevant functions for net connections.
<a href="#">Serial Ports</a>	Communicating with serial ports.
<a href="#">Byte Packing</a>	Using bindat to pack and unpack binary data.

### Receiving Output from Processes

<a href="#">Process Buffers</a>	By default, output is put in a buffer.
<a href="#">Filter Functions</a>	Filter functions accept output from the process.
<a href="#">Decoding Output</a>	Filters can get unibyte or multibyte strings.
<a href="#">Accepting Output</a>	How to wait until process output arrives.

### Low-Level Network Access

<a href="#">Network Processes</a>	Using make-network-process.
<a href="#">Network Options</a>	Further control over network connections.
<a href="#">Network Feature Testing</a>	Determining which network features work on the machine you are using.

### Packing and Unpacking Byte Arrays

<a href="#">Bindat Spec</a>	Describing data layout.
<a href="#">Bindat Functions</a>	Doing the unpacking and packing.
<a href="#">Bindat Examples</a>	Samples of what bindat.el can do for you!

### Emacs Display

<a href="#">Refresh Screen</a>	Clearing the screen and redrawing everything on it.
<a href="#">Forcing Redisplay</a>	Forcing redisplay.
<a href="#">Truncation</a>	Folding or wrapping long text lines.
<a href="#">The Echo Area</a>	Displaying messages at the bottom of the screen.
<a href="#">Warnings</a>	Displaying warning messages for the user.
<a href="#">Invisible Text</a>	Hiding part of the buffer text.
<a href="#">Selective Display</a>	Hiding part of the buffer text (the old way).
<a href="#">Temporary Displays</a>	Displays that go away automatically.
<a href="#">Overlays</a>	Use overlays to highlight parts of the buffer.
<a href="#">Size of Displayed Text</a>	How large displayed text is.

<a href="#">Line Height</a>	Controlling the height of lines.
<a href="#">Faces</a>	A face defines a graphics style for text characters: font, colors, etc.
<a href="#">Fringes</a>	Controlling window fringes.
<a href="#">Scroll Bars</a>	Controlling scroll bars.
<a href="#">Window Dividers</a>	Separating windows visually.
<a href="#">Display Property</a>	Enabling special display features.
<a href="#">Images</a>	Displaying images in Emacs buffers.
<a href="#">Buttons</a>	Adding clickable buttons to Emacs buffers.
<a href="#">Abstract Display</a>	Emacs's Widget for Object Collections.
<a href="#">Blinking</a>	How Emacs shows the matching open parenthesis.
<a href="#">Character Display</a>	How Emacs displays individual characters.
<a href="#">Beeping</a>	Audible signal to the user.
<a href="#">Window Systems</a>	Which window system is being used.
<a href="#">Tooltips</a>	Tooltip display in Emacs.
<a href="#">Bidirectional Display</a>	Display of bidirectional scripts, such as Arabic and Farsi.
<b>The Echo Area</b>	
<a href="#">Displaying Messages</a>	Explicitly displaying text in the echo area.
<a href="#">Progress</a>	Informing user about progress of a long operation.
<a href="#">Logging Messages</a>	Echo area messages are logged for the user.
<a href="#">Echo Area Customization</a>	Controlling the echo area.
<b>Reporting Warnings</b>	
<a href="#">Warning Basics</a>	Warnings concepts and functions to report them.
<a href="#">Warning Variables</a>	Variables programs bind to customize their warnings.
<a href="#">Warning Options</a>	Variables users set to control display of warnings.
<a href="#">Delayed Warnings</a>	Deferring a warning until the end of a command.
<b>Overlays</b>	
<a href="#">Managing Overlays</a>	Creating and moving overlays.
<a href="#">Overlay Properties</a>	How to read and set properties. What properties do to the screen display.
<a href="#">Finding Overlays</a>	Searching for overlays.
<b>Faces</b>	
<a href="#">Face Attributes</a>	What is in a face?
<a href="#">Defining Faces</a>	How to define a face.
<a href="#">Attribute Functions</a>	Functions to examine and set face attributes.
<a href="#">Displaying Faces</a>	How Emacs combines the faces specified for a character.
<a href="#">Face Remapping</a>	Remapping faces to alternative definitions.
<a href="#">Face Functions</a>	How to define and examine faces.
<a href="#">Auto Faces</a>	Hook for automatic face assignment.
<a href="#">Basic Faces</a>	Faces that are defined by default.
<a href="#">Font Selection</a>	Finding the best available font for a face.

<a href="#">Font Lookup</a>	Looking up the names of available fonts and information about them.
<a href="#">Fontsets</a>	A fontset is a collection of fonts that handle a range of character sets.
<a href="#">Low-Level Font</a>	Lisp representation for character display fonts.
<b>Fringes</b>	
<a href="#">Fringe Size/Pos</a>	Specifying where to put the window fringes.
<a href="#">Fringe Indicators</a>	Displaying indicator icons in the window fringes.
<a href="#">Fringe Cursors</a>	Displaying cursors in the right fringe.
<a href="#">Fringe Bitmaps</a>	Specifying bitmaps for fringe indicators.
<a href="#">Customizing Bitmaps</a>	Specifying your own bitmaps to use in the fringes.
<a href="#">Overlay Arrow</a>	Display of an arrow to indicate position.
<b>The display Property</b>	
<a href="#">Replacing Specs</a>	Display specs that replace the text.
<a href="#">Specified Space</a>	Displaying one space with a specified width.
<a href="#">Pixel Specification</a>	Specifying space width or height in pixels.
<a href="#">Other Display Specs</a>	Displaying an image; adjusting the height, spacing, and other properties of text.
<a href="#">Display Margins</a>	Displaying text or images to the side of the main text.
<b>Images</b>	
<a href="#">Image Formats</a>	Supported image formats.
<a href="#">Image Descriptors</a>	How to specify an image for use in <code>:display</code> .
<a href="#">XBM Images</a>	Special features for XBM format.
<a href="#">XPM Images</a>	Special features for XPM format.
<a href="#">PostScript Images</a>	Special features for PostScript format.
<a href="#">ImageMagick Images</a>	Special features available through ImageMagick.
<a href="#">Other Image Types</a>	Various other formats are supported.
<a href="#">Defining Images</a>	Convenient ways to define an image for later use.
<a href="#">Showing Images</a>	Convenient ways to display an image once it is defined.
<a href="#">Multi-Frame Images</a>	Some images contain more than one frame.
<a href="#">Image Cache</a>	Internal mechanisms of image display.
<b>Buttons</b>	
<a href="#">Button Properties</a>	Button properties with special meanings.
<a href="#">Button Types</a>	Defining common properties for classes of buttons.
<a href="#">Making Buttons</a>	Adding buttons to Emacs buffers.
<a href="#">Manipulating Buttons</a>	Getting and setting properties of buttons.
<a href="#">Button Buffer Commands</a>	Buffer-wide commands and bindings for buttons.
<b>Abstract Display</b>	
<a href="#">Abstract Display Functions</a>	Functions in the Ewoc package.
<a href="#">Abstract Display Example</a>	Example of using Ewoc.

## Character Display

<a href="#">Usual Display</a>	The usual conventions for displaying characters.
<a href="#">Display Tables</a>	What a display table consists of.
<a href="#">Active Display Table</a>	How Emacs selects a display table to use.
<a href="#">Glyphs</a>	How to define a glyph, and what glyphs mean.
<a href="#">Glyphless Chars</a>	How glyphless characters are drawn.

## Operating System Interface

<a href="#">Starting Up</a>	Customizing Emacs startup processing.
<a href="#">Getting Out</a>	How exiting works (permanent or temporary).
<a href="#">System Environment</a>	Distinguish the name and kind of system.
<a href="#">User Identification</a>	Finding the name and user id of the user.
<a href="#">Time of Day</a>	Getting the current time.
<a href="#">Time Conversion</a>	Converting a time from numeric form to calendrical data and vice versa.
<a href="#">Time Parsing</a>	Converting a time from numeric form to text and vice versa.
<a href="#">Processor Run Time</a>	Getting the run time used by Emacs.
<a href="#">Time Calculations</a>	Adding, subtracting, comparing times, etc.
<a href="#">Timers</a>	Setting a timer to call a function at a certain time.
<a href="#">Idle Timers</a>	Setting a timer to call a function when Emacs has been idle for a certain length of time.
<a href="#">Terminal Input</a>	Accessing and recording terminal input.
<a href="#">Terminal Output</a>	Controlling and recording terminal output.
<a href="#">Sound Output</a>	Playing sounds on the computer's speaker.
<a href="#">X11 Keysyms</a>	Operating on key symbols for X Windows.
<a href="#">Batch Mode</a>	Running Emacs without terminal interaction.
<a href="#">Session Management</a>	Saving and restoring state with X Session Management.
<a href="#">Desktop Notifications</a>	Desktop notifications.
<a href="#">File Notifications</a>	File notifications.
<a href="#">Dynamic Libraries</a>	On-demand loading of support libraries.
<a href="#">Security Considerations</a>	Running Emacs in an unfriendly environment.

## Starting Up Emacs

<a href="#">Startup Summary</a>	Sequence of actions Emacs performs at startup.
<a href="#">Init File</a>	Details on reading the init file.
<a href="#">Terminal-Specific</a>	How the terminal-specific Lisp file is read.
<a href="#">Command-Line Arguments</a>	How command-line arguments are processed, and how you can customize them.

## Getting Out of Emacs

<a href="#">Killing Emacs</a>	Exiting Emacs irreversibly.
<a href="#">Suspending Emacs</a>	Exiting Emacs reversibly.

## Terminal Input

<a href="#">Input Modes</a>	Options for how input is processed.
<a href="#">Recording Input</a>	Saving histories of recent or all input events.
<b>Preparing Lisp code for distribution</b>	
<a href="#">Packaging Basics</a>	The basic concepts of Emacs Lisp packages.
<a href="#">Simple Packages</a>	How to package a single .el file.
<a href="#">Multi-file Packages</a>	How to package multiple files.
<a href="#">Package Archives</a>	Maintaining package archives.

## Tips and Conventions

<a href="#">Coding Conventions</a>	Conventions for clean and robust programs.
<a href="#">Key Binding Conventions</a>	Which keys should be bound by which programs.
<a href="#">Programming Tips</a>	Making Emacs code fit smoothly in Emacs.
<a href="#">Compilation Tips</a>	Making compiled code run fast.
<a href="#">Warning Tips</a>	Turning off compiler warnings.
<a href="#">Documentation Tips</a>	Writing readable documentation strings.
<a href="#">Comment Tips</a>	Conventions for writing comments.
<a href="#">Library Headers</a>	Standard headers for library packages.

## GNU Emacs Internals

<a href="#">Building Emacs</a>	How the dumped Emacs is made.
<a href="#">Pure Storage</a>	Kludge to make preloaded Lisp functions shareable.
<a href="#">Garbage Collection</a>	Reclaiming space for Lisp objects no longer used.
<a href="#">Stack-allocated Objects</a>	Temporary conses and strings on C stack.
<a href="#">Memory Usage</a>	Info about total size of Lisp objects made so far.
<a href="#">C Dialect</a>	What C variant Emacs is written in.
<a href="#">Writing Emacs Primitives</a>	Writing C code for Emacs.
<a href="#">Object Internals</a>	Data formats of buffers, windows, processes.
<a href="#">C Integer Types</a>	How C integer types are used inside Emacs.

## Object Internals

<a href="#">Buffer Internals</a>	Components of a buffer structure.
<a href="#">Window Internals</a>	Components of a window structure.
<a href="#">Process Internals</a>	Components of a process structure.

Copyright © 1990–1996, 1998–2017 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with the Invariant Sections being “GNU General Public License,” with the Front-Cover Texts being “A GNU Manual,” and with the Back-Cover Texts as in (a) below. A copy of the license is included in the section entitled “GNU Free Documentation License.”



(a) The FSF's Back-Cover Text is: “You have the freedom to copy and modify this GNU manual. Buying copies from the FSF supports it in developing GNU and promoting software freedom.”



