function definitions, whitespace, and more. You can use them anywhere you read or edit text: source code, web pages, shells, directory listings, email messages, and so on.

Moving around in buffers

The most basic buffer movement commands move point (the cursor) by rows (lines) or columns (characters):

`C-f` Forward one character
`C-n` Next line
`C-b` Back one character
`C-p` Previous line

Here are some ways to move around in larger increments:

`C-a` Beginning of line
`M-f` Forward one word
`M-a` Previous sentence
`M-v` Previous screen
`M-<` Beginning of buffer
`C-e` End of line
`M-b` Back one word
`M-e` Next sentence
`C-v` Next screen
`M->` End of buffer

When you get used to these keys, they're faster than their more familiar equivalents in other applications (Home, End, Ctrl+Left, etc.) because you don't have to move your hands from the touch typing position. And these keys are far faster than using a mouse to move around in a buffer.

Emacs often provides additional commands for moving around in context-specific ways (e.g. in source code, commands to move to the previous or next function definition).

Many of the above commands move to a location relative to your current position in the buffer, so you can use them repeatedly (e.g. `C-p C-p` `C-p` to move back three lines). You can use the **prefix argument** to save time: `C-u` followed by a number and a movement command repeats that command the specified number of times. You can also use `M-[digit]` instead of `C-u [digit]`. If you use `C-u` without specifying a digit, the default is 4. Omitting the digit can save time when you don't know precisely how many units you want to jump anyway.

`C-u 3 C-p` Back 3 lines
`C-u 10 C-f` Forward 10 characters
`M-1 M-0 C-f` Forward 10 characters
`C-u C-n` Forward 4 lines
`C-u C-u C-n` Forward 16 lines
`C-u C-u C-u C-n` Forward 64 lines

You can jump directly to a particular line number in a buffer:

`M-g g` Jump to specified line

Searching for text is a handy way to move around in a buffer. Think of search as just another facility for movement. When you're looking for something specific, you can use incremental search to take you right there instead of scanning by lines or pages. More about search later.

`C-s` Incremental search forward

`C-r` Incremental search backward

One other way of moving around in buffers is by using the **mark**:

Mark

Emacs remembers something called the **mark**, which is a previous cursor position. You can set mark to indicate a particular place in your buffer so you can return to it easily. `C-x C-x` at a later time will return point to mark. Actually, that command also moves mark to where point formerly was; therefore, a second `C-x C-x` returns point to its original position.

`C-SPC` Set mark to the current location

`C-x C-x` Swap point and mark

You can set mark explicitly, but certain commands set mark for you, providing you with convenient footholds as you move around your buffer:

When you...	mark is set to...
Type `C-SPC`	your current location
Jump to either end of the buffer (`M-<` or `M->`)	your previous location
Exit incremental search	where you began searching
Yank text	the beginning of the yanked region
Insert a buffer or file	the beginning of the inserted text

As you can see, Emacs tries to be helpful: many commands that have the potential to take you long distances set mark so that a simple `C-x C-x` takes you back to where you were. **Emacs makes it difficult to lose your place in a buffer**: even if you take a detour, you don't need to scroll around to get back to where you were.

Emacs saves many previous values of the mark for you. You can cycle through the **mark ring**, which contains the last 16 marks you've set in the current buffer:

`C-u C-SPC` Cycle through mark ring

Region

Mark serves another purpose: mark and point together delineate **the region**. Many commands operate only on the text in the region (i.e. between mark and point). You can set the region explicitly by setting mark (`C-SPC`) and then moving point elsewhere, or by clicking and dragging with the mouse. Emacs provides some commands which set the region for you by moving point and mark appropriately, for example:

`C-x h` Make region contain the entire buffer ("Select all")

`M-h` Make region contain the current paragraph

Other commands helpfully set the region as part of what they do. `C-y` (yank), inserting a file, and inserting a buffer all set the region to surround the inserted text.

Narrowing restricts the view (and editing) of a buffer to a certain region. This is handy when you're only working with a small part of a buffer (e.g. a chapter in a book). Then commands like incremental search, or `beginning-of-buffer` or `end-of-buffer` don't lead you out of the region of interest, and commands like search and replacement don't affect the entire file.

`C-x n n` Narrow buffer to the current region

`C-x n w` Restore ("widen") buffer

For more information see `(info "(emacs)Narrowing")`.

Killing ("cutting") text

As with text movement, Emacs provides commands for deleting text in various amounts.

`C-k` kills the portion of the current line after point (or deletes the newline following point if point is at the end of line). The prefix argument for `C-k` can be used to kill multiple lines:

`C-k` Kill line
`C-u 10 C-k` Kill 10 lines
The following commands operate on the region, and are the closest analogs to "cut" and "copy" in Emacs:

`C-w` Kill region ("cut")
`M-w` Save region to kill ring without deleting ("copy")
These commands are also handy:

`M-d` Kill next word
`M-k` Kill to end of sentence
All of the above commands **kill** the text being deleted, which means that Emacs removes the text and squirrels it away for later retrieval ("yanking"). Most commands which delete significant amounts of text kill it rather than simply removing it, so that you can use those commands either to "delete" text or to "cut" it for later use.

Yanking ("pasting") text

After a piece of text has been killed, it goes to a place called the **kill ring** which is analagous to the "clipboard": you can **yank** an item to restore it from the kill ring with `C-y`. Unlike the clipboard, however, the kill ring is capable of holding many different items. If the item you want to yank is not placed when you type `C-y`, type `M-y` (repeatedly, if necessary) to cycle through previously killed items.

`C-y` Yanks last killed text
`M-y` Replace yanked text with previously killed text
Recall that most commands which delete a large amount of text in fact kill it (i.e. place it in the kill ring) so you can restore it later. **Emacs makes it very difficult to lose a lot of text permanently**: in editors with only a single clipboard, one can easily accidentally delete a large chunk of text or clobber the contents of the clipboard (by cutting two items in succession). But in Emacs, in either of those cases, the lost text can easily be retrieved from the kill ring.

Undo

Emacs' undo facility works slightly differently from that of other editors. In most editors, if you undo some changes, then make some new changes, the states formerly accessible with "redo" can no longer be recovered! So when using "undo" and "redo" extensively, one has to be very careful to avoid accidentally clobbering the redo list.

Emacs uses a different undo model which does not have this deficiency. After any consecutive sequence of undos, Emacs makes all your previous actions undoable, including the undos. (This will happen whenever a sequence of undos is broken by any other command.)

If this sounds complicated, just remember that "undo" is always capable of getting you back to **any previous state your buffer was in** (unless Emacs has run out of memory to store the undo history). The principle here is that **Emacs makes it very difficult to accidentally lose your work**.

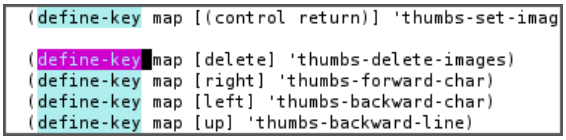
Undo is available via three different keys:

`C-/` Undo
`C-_` Undo
`C-x u` Undo
So if you need to get back to a previous buffer state, simply move the cursor (so as to break any existing sequence of undos), and press `C-/` until you find what you want.

To learn more about undo, see `(info "(emacs)Undo")`.

Incremental search

`C-s` Incremental search
Typing `C-s` followed by some text starts incremental search. Emacs jumps to the next occurrence of whatever you have typed, as you are typing it (you may have seen similar behavior in Mozilla Firefox or other web browsers), and all matches visible on your screen are highlighted.



Within incremental search, you can type `C-s` again at any time to jump to the next occurrence.

When you've found what you're looking for, you can either type `RET` (or use almost any movement command) to exit search at the occurrence you've found, or `C-g` ("cancel") to return to where your search started. If you exit search at the found occurrence, you can easily jump back to where you started with `C-x C-x` since incremental search sets mark appropriately.

These commands help you to issue previously issued queries:

`C-s C-s` Search for most recently searched item
`C-s M-p` Previous item in search history
`C-s M-n` Next item in search history
`C-h k C-s` Guide to more commands available in incremental search
You can perform a backward incremental search with `C-r`. (All the above commands can be activated similarly from within backward search.) At any time during a forward (or backward) search, you can type `C-r` (`C-s`) to switch to a backward (forward) search.

`C-r` Backward incremental search
See `(info "(emacs)Incremental Search")` for more information.

Search and replacement

`M-%` Query replace
The query replace command prompts you for a search string and a replacement. Then, for each match in the buffer, you can choose whether or not to replace the search string. Here are some of the options available at each prompt:

- Type `y` to replace the current match.
- Type `n` to skip to the next match without replacing.

- Type `q` to exit without doing any more replacements.
- Type `.` to replace this match, then exit.
- Type `!` to replace all remaining matches with no more questions.

See `(info "(emacs)Query Replace")` for more information about these (and other) options. You can also type `?` anytime inside a search-and-replace operation to see a guide.

Regular expression search

Emacs allows you to search for regular expressions:

`C-M-s` Regular expression incremental search

Regular expressions are a succinct way of searching for many different strings at once by using a special language to describe the form of what you're looking for. Regular expression syntax is beyond the scope of this tour; see `(info "(emacs)Regexps")` for more information.

If you're new to regexps, or you are constructing a particularly complicated regexp, you can use the regexp builder (`M-x re-builder`). This command pops up a separate window in which you can test out your regexp, and any matches in your original buffer will get highlighted as you edit your regexp.

Instead of jumping through matches one by one, you can also choose to display them all at once. `M-x occur` prompts you for a regular expression, then displays in a separate buffer a list of all lines in the current buffer which match that regexp (as well as their line numbers). Clicking on any occurrence takes you to that line in the buffer.

Regular expression search and replacement

Regular expressions are even more powerful in search and replace, because Emacs allows the replacement text to depend on the found text. You can control replacement by inserting special escape sequences in the replacement string, and Emacs will substitute them appropriately:

When you type this in a replacement string:	Emacs replaces it with:
`\&`	the original found text
`\1`, `\2`, etc.	the 1st, 2nd, etc. parenthesized subgroup in the found text
`\#`	the number of replacements done so far
`\?`	a string obtained by prompting the user on each match
`\,(lisp-expression …)`	the result of evaluating an arbitrary function

Here's an example. Suppose we have a buffer containing names like this:

```
George Washington
John Adams
Thomas Jefferson
James Madison
James Monroe
```

If we run `M-x replace-regexp` and replace the regexp `\(\w+\) \(\w+\)` with `\,(upcase \2), \1`, our buffer now looks like this:

```
WASHINGTON, George
ADAMS, John
JEFFERSON, Thomas
MADISON, James
MONROE, James
```

As you can see, regexp replacement is capable of doing some pretty sophisticated transformations. (Roughly, the search expression searches for two words; the replacement string inserts an uppercased version of the second word, followed by a comma, followed by the first word.)

Keyboard Macros

Keyboard macros are a way to remember a fixed sequence of keys for later repetition. They're handy for automating some boring editing tasks.

`F3` Start recording macro
`F4` Stop recording macro
`F4` Play back macro once
`M-5 F4` Play back macro 5 times
`M-0 F4` Play back macro over and over until it fails
For example, this sequence of keys does the exact same transformation that we did with regular expression replacement earlier, that is, it transforms a line containing `George Washington` to `WASHINGTON, George`:

`M-d C-d M-u, [SPC] C-y C-n C-a`

After we record that key sequence as a macro, we can type `M-0 F4` to transform the buffer pictured earlier; in this case, Emacs runs the macro repeatedly until it has reached the end of the buffer.

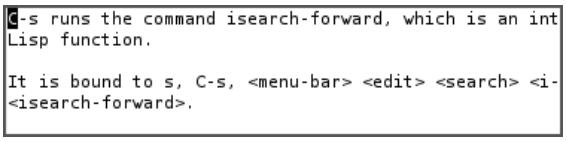
See `(info "(emacs)Keyboard Macros")` for more information.

Help with commands

If you've read this far, you are probably intimidated by the thought of having to remember a bunch of keyboard commands and command names. Fortunately, Emacs includes comprehensive and easily accessible documentation. The documentation isn't just for beginners. Emacs has thousands of commands, of which most people only use a small number. So even Emacs experts frequently consult the docs in order to learn about new commands or jog their memory on old ones.

If you don't remember what a particular key or command does, you can read a description of it by using one of the following commands:

`C-h k`
Shows documentation for the command associated with any particular key.
`C-h f`
Shows documentation for any particular command, by name (i.e. what you would type after `M-x`).
For example, `C-h k C-s` and `C-h f isearch-forward RET` both display a page describing incremental search:



This is handy, for example, if you don't remember what `C-s` does, or if you remember that it invokes incremental search but want to know more about that feature. The documentation gives the full name of the command, shows which (if any) keys are bound to it, and gives a complete description of what the command does.

On the other hand, if you don't remember how to invoke a particular feature, you can use **apropos** to search for it:

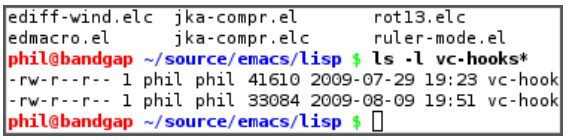
`C-h a`
Search for commands by keywords or regexp
For example, if I remember that I want to activate **narrowing**, but don't remember how, I can type `C-h a narrow RET` which shows a brief list of commands having to do with **narrow**, one of which is `M-x narrow-to-region`.

More useful features

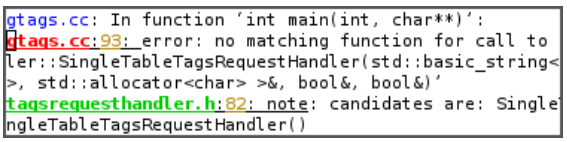
Integration with common tools

Emacs is notable for its integration with many common tools. Not only can you invoke them from within the editor, Emacs usually helps you use their output more effectively. Here are some examples:

`M-x shell`
Starts a shell in the buffer named `*shell*`, switching to it if it already exists. Use `C-u M-x shell` to use a buffer with a different name.



`M-x compile`
Invokes `make` (with targets and options of your choice) and displays output in a new buffer. Emacs identifies error lines containing filenames and line numbers, and you can click on them to jump directly to the corresponding buffer and line.



`M-x gdb`
Invokes `gdb` in a new buffer. You can use the gdb command line as usual in that buffer. However, Emacs lets you set breakpoints directly from your source buffers and shows execution by marking the active line in your source buffers. Emacs can also display breakpoints, the stack, and locals, simultaneously and each in their own window.