Emacs

Prominent UNIX Editors

There have been four prominent and widely popular UNIX editors.

ed is the original UNIX editor. It is line-oriented and terse, but elegant. ed, or a lookalike, is on most UNIX systems.

vi was created by Bill Joy in 1976. It is screen oriented and "modal". It has a second "personality", called ex, that is essentially an improved version of ed. Arguably the fastest plain-text editor for touch-typists.

In 1981, James Gosling created "UNIX Emacs", a C implementation that was similar to Richard Stallman's Emacs for the PDP-10. Important difference: Gosling's version provided "Mock Lisp", not a "true" Lisp.

In 1984-1985 Stallman created GNU Emacs—the first tangible result of the GNU project.

The core of GNU Emacs is written in C but it contains a "true" Lisp that in turn is used to implement much of the editor's functionality.

GNU Emacs is the Emacs that we'll be using.

Originally, "Emacs" was an acronym for Editor MACroS.

Today, Emacs is billed as "The extensible self-documenting text editor".

Why use Emacs?

What's good about Emacs?

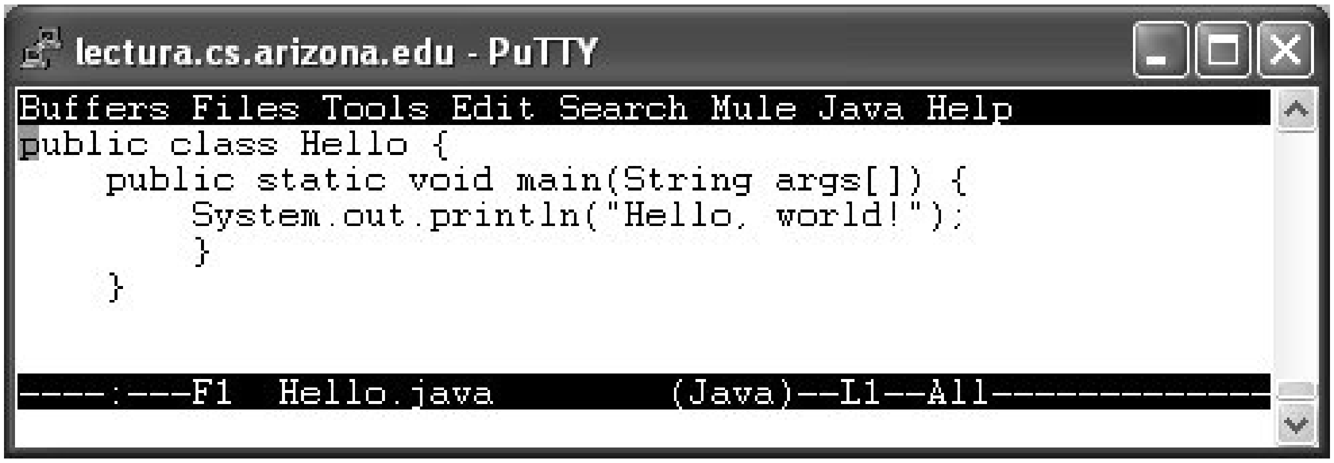
- Widely available on UNIX and Windows
- Free
- Multiple files; multiple "windows" per file
- Fully customizable
- Programmable (via Lisp)
- Lots of existing "packages"

What's bad about Emacs?

- Not a state-of-the-art IDE
- Complex
- Control-ALT-Shift-Cokebottle
- Packages are of varying quality
- Undisciplined accretion of commands

Running Emacs

To start Emacs on lectura, use 'emacs *filename*'. When Emacs starts, it fills the window of the ssh client:


lectura.cs.arizona.edu - PuTTY
Buffers Files Tools Edit Search Mule Java Help

```
public class Hello {
    public static void main(String args[]) {
        System.out.println("Hello, world!");
        }
    }

----:---F1  Hello.java       (Java)--L1--All-------------
```

The contents of the file are loaded into a *buffer*, which is displayed in an Emacs *window*.

A "mode line" associated with the buffer appears below it. We can see that the buffer holds Hello.java, we're in Java mode, the cursor is on line 1, and all of the file is visible.

Below the mode line is the *minibuffer*. It displays status messages at various times. It is also used for some interactions with Emacs.

The "menu bar" at the top of the window is awkward to use in non-windowed mode.

A no-frills tutorial can be started by typing C-h t. (Ctrl+h, then t.)

To terminate Emacs, type C-x C-c (^X^C).

Basics of interaction with Emacs

As a rule, characters that are typed are inserted into the current buffer at the cursor position.

Operations such as moving around in a line, deleting text, and scrolling the text in view are usually invoked using a sequence of keystrokes beginning with the control or Alt keys, or a function key.

Here are examples of the notation for key sequences:

> C-x indicates Ctrl+x (^X)
>
> M-x indicates Alt+x or ESC, then x
>
> C-x b indicates Ctrl+x, then b
>
> M-a C-b c D indicates Alt+a (or ESC, then a), then Ctrl+b, then c, then Shift+D

A little about M-x:

> The "M" in "M-x" stands for "Meta". Some keyboards once used at MIT and elsewhere had additional shift keys. Meta was one of them.
>
> The ALT key on PC keyboards can be used as a Meta-like shift key (in Emacs).
>
> An alternative to using ALT+x is to type ESC and then type x—two separate keystrokes.

Sidebar: SSH Secure Shell and Emacs

PuTTY doesn't require any special measures to work well with Emacs but there are two things to note when using SSH Secure Shell:

(1) Some versions of SSH S.S. require that a setting be adjusted to cause the Alt key to function as a Meta key.

To have Alt work as Meta key, chose Edit | Settings ... Keyboard and activate "Use Alt as meta key" (in SSH Secure Shell, not Emacs).

(2) By default, SSH S.S. transmits a C-h when the backspace key is pressed. By default, C-h starts the Emacs help system, instead of erasing the last character.

If you wish to use backspace to erase characters, choose Edit|Settings...Keyboard and activate "Backspace sends Delete". (Alternative: Use the Del key to erase characters.)

The instructor strongly recommends both settings.

Basics of interaction, continued

Keystroke sequences are bound to commands. Examples:

- C-b is bound is to the command backward-char
- C-x C-b is bound to list-buffers
- C-x b is bound to switch-to-buffer
- ESC C-v is bound to scroll-other-window
- C-h b is bound to describe-bindings
- C-h k is bound to describe-key
- a is bound to self-insert-command
- RET is bound to the command newline

The full set of keystroke sequences forms a tree. The tree is walked as keys are typed. When a terminal node of the tree is reached, the command associated with that node is executed.

If the user pauses when typing, Emacs shows the key sequence in progress.

C-g interrupts a key sequence in progress.

Key bindings can be changed but this material covers the standard set of bindings.

<u>Some of the standard bindings make more sense than others.</u>

Basic navigation and editing

The current position in the buffer is known as "the point". The point is actually <u>between</u> characters but the cursor is shown on the character following the point.

Typing ordinary characters such as a, 9, and $ inserts them at the point.

Operations such as character-by-character and "word"-by-word movement and deletion are indicated with C- and M- keys:

	Move Forward	Move Backward	Delete Forward	Delete Backward
Character	C-f	C-b	C-d	DEL
"Word"	M-f	M-b	M-d	M-DEL

A data structure known as a "syntax table" specifies what constitutes a "word".

In some cases, PC keyboard keys behave as in typical Windows applications. For example, the left arrow and right arrow keys move back and forth between characters.

Note that the PC keyboard backspace key is considered by Emacs to be the DEL key.

Basic navigation and editing, continued

Here are some very common operations and bindings:

- C-n moves down one line, C-p moves up a line.
- C-a positions the point at the start of the current line, C-e at the end.
- M-< positions the point at the beginning of the buffer, M-> at the end. Remember: ESC < is equivalent to M-<.
- M-v moves the point up by a screen, C-v moves down by a screen.
- C-g is the Emacs "interrupt" key. Use it to abandon a key sequence in progress.
- **C-x C-s saves the contents of the current "buffer".**

The undo command, bound to C-_ (underscore), undoes changes to buffer contents. Any number of changes may be undone. Undos may be undone.

Interrupting a series of undos with C-g (or any operation except undo) and then proceeding with more undos will cause the undos to be undone.

Killing and yanking

C-k is bound to the command kill-line. It "kills" the text between the point and the newline character at the end of the current line.

When text is killed, it is put in the "kill ring".

The most recent kill can be "yanked" with C-y. The killed text is inserted at the point. The most recent kill can be yanked any number of times.

Consecutive kills accumulate in a single kill ring entry.

Example:

> The sequence C-a C-k C-k C-k C-k C-y C-y C-y deletes two lines and then inserts three copies of those two lines.

M-d and M-DEL are also kills (kill-word and backward-kill-word) and create (or append to) kill ring entries.

The yank-pop command, bound to M-y, replaces a just-yanked portion of text with the previous kill. (Try this: Do three non-consecutive kills, then cycle between them with M-y three times.)

The mark and the region

One type of object in the Emacs Lisp system is a *marker*. A marker specifies a position in a buffer but a marker is more than just an integer. Markers are "sticky"—surrounding text can change but a marker stays between the characters where it was originally placed.

Each buffer has a distinguished marker called "the mark" that is used by various user-level commands.

set-mark-command, bound to C-@ (and C-space), sets the mark.

<u>The text between the point and the mark is known as "the region".</u>

A number of commands operate on the region. Examples:

- kill-region (C-w) kills the region (and puts it in the kill-ring).
- write-region (not bound) writes the region to a file.
- upcase-region (C-x C-u) converts characters in the region to upper case.
- copy-region-as-kill (not bound) copies the region in the kill-ring, but doesn't kill it.
- tibetan-compose-region (not bound) makes composite chars from Tibetan character components in the region

The mark and the region, continued

C-x C-x swaps the point and the mark. It might be used to double-check that the region selected is as desired, and/or make adjustments in the other end.

Unfortunately, when running in terminal mode there is no visible indication of the extent of the region.

Some commands, such as beginning-of-buffer (M-<) set the mark as a side-effect. If so, "Mark Set" is displayed in the minibuffer.

narrow-to-region (C-x n n) hides the text outside the region. widen (C-x n w) removes the restriction. One application of narrowing is to restrict searches to a single routine.

universal-argument

The universal-argument command is bound to C-u. It is used to pass a numeric argument to a command.

Two examples:

- C-u 2 0 C-n moves the cursor down twenty lines.
- M-< C-u 4 9 C-f positions the cursor on the 50th character in the buffer.

If no numeric keys follow C-u, an argument of 4 is passed. Successive C-u keys produces powers of 4. Examples:

- C-u C-p moves the cursor up four lines.
- C-u C-u C-d deletes the next sixteen characters.
- C-u C-u C-u C-u C-u C-u x inserts 4096 "x"s

An argument of N does not necessarily cause N repetitions of a command.

Example: C-u C-k kills four lines, but C-k C-k C-k C-k kills two!

Speculate: What would C-u C-y do? (Recall that C-y is yank.)

An argument can be negative: C-u - 1 5 C-f moves the point fifteen characters to the left.

A numeric argument is sometimes called a *prefix argument*.

Searching and replacing

Emacs provides *incremental searching*—as characters are typed the accumulated string is used as the subject of the search.

An incremental search is initiated with C-s. With each subsequent keystroke the cursor advances to the first occurrence of the accumulated string, starting at the point.

While a search is active, i.e. while the "I-search:" prompt is displayed in the minibuffer, the user has several choices in addition to extending the current search string with additional characters. Here are some of the possibilities:

- DEL removes the last-typed character from the current string and reverts the point to its position before that character was typed.

- C-s searches for the next occurrence of the current string; C-r searches for the previous occurrence.

- RET terminates the search and leaves the cursor (the point) at its present position.

- C-g terminates the search and reverts the point to its position before the search started.

A keystroke that doesn't have meaning in incremental search mode terminates the search just as RET does.

The sequence C-s C-s starts an incremental search using the last search string as the current string.

Searching and replacing, continued

query-replace (M-%) first prompts for a search string and a replacement, and then, on each match, prompts for an action.

Non-incremental searching can be done with search-forward and search-backward but they provide no easy way to repeat the search.

Searches using "regular expressions" are supported with re-search-forward, re-search-backward, isearch-backward-regexp, and isearch-forward-regexp.

"Windows"

The Emacs screen can be split into two or more tiled "windows". Each window has a text area and a mode line.

split-window-vertically (C-x 2) divides the current window into two vertically tiled windows. A window resulting from a split can be split again, subject to minimum size constraints.

When a window is split, the current buffer is displayed in both windows. Each window has its own point.

other-window (C-x o) cycles the cursor through each window in turn.

delete-window (C-x 0) (zero) deletes the current window (the window that contains the cursor).

delete-other-windows (C-x 1) deletes all windows except the current window.

split-window-horizontally (C-x 3) divides the current window into two horizontally-tiled windows.

A few commands, such as scroll-other-window (M-C-v) and scroll-other-window-down (M-C-V) (ouch!) operate on the "next" window.

Opening files with find-file

To open files in an already-running Emacs, use find-file (C-x C-f).

Typing C-x C-f produces a "Find file:" prompt (in the minibuffer). Responding with a question mark shows alternatives; TAB can be used to complete file names.

If the file exists, it is loaded and displayed. If it does not exist, it will be created upon the first save.

Opening a file is sometimes called "visiting" a file.

It is common to start Emacs without naming a file and then use find-file as needed to open files of interest.

If you mistype a file name you can use find-alternate-file to open another file and as a side effect, close the previously opened file.

IMPORTANT note on find-file: if a directory is named then the "directory editor" dired, is started. A q safely terminates dired.

Buffers

Buffers hold text that is being edited.

When a file is opened with find-file, Emacs creates a buffer, loads the contents of the file into the buffer, and then creates a window that displays the buffer contents.

Zero or more windows are associated with each buffer. A window never exists without a buffer. A buffer typically has a file associated with it, but is that not required.

If Emacs is started with no files, the buffer *scratch* is displayed. No file is associated with *scratch*.

list-buffers (C-x C-b) creates a buffer named *Buffer List* that displays a variety of information about existing buffers.

switch-to-buffer (C-x b) prompts for buffer name to switch to. As with find-file, ? displays alternatives and TAB completes names. If the buffer specified does not exist, it is created.

kill-buffer (C-x k) queries for a buffer and destroys it.

revert-buffer reloads the buffer with the current on-disk contents of the file.

Help (!)

Emacs has extensive built-in documentation.

Typing C-h C-h creates a buffer named *Help* that displays a number of help options. Here are some of them:

- b Runs describe-bindings. It displays the current set of key bindings.

- k Runs describe-key. It prompts for a key sequence and shows which command, if any, the key sequence is bound to.

- f Runs describe-function. It prompts for a command (or function) name and displays a description of the function.

- a Runs command-apropos. It prompts for a string and displays a list of commands that contain the string.

  Example: To see all buffer-related commands, enter "buffer" at the "Apropos command (regexp):" prompt.

- w Runs where-is. It prompts for the name of a command and tells what key sequence the command is bound to, if any.

- i Starts the "Info" documentation reader.

Help, continued

In most cases the result of a help command is a buffer named *Help*, which can be treated like any other buffer. (C-x o to switch to the window, C-x 0 (zero) to close it, M-C-v to scroll it down, M-C-V to scroll it up.)

A help option can be accessed directly by following C-h with the appropriate letter. For example, C-h k runs describe-key.

In the *Help* buffer, further help can be often be obtained by positioning the cursor on an element such as a command name and pressing RET.

Info

Emacs contains a text-based documentation reader/browser called "Info".

Info has documentation on Emacs itself, Emacs packages, and various GNU utilities.

Additionally, any user can create documentation that is browsable by Info.

Info is started with C-h i. Keystrokes are used to navigate through the material, which is tree-structured.

Simple navigation can be performed using the arrow keys.

Example:

> On the initial Info screen, moving down to the line "* Emacs: (emacs)" and then RET goes to the Emacs "node" of the Info tree.

Alternatively, a node by reached by typing m, which produces a prompt for the name of a menu item. Item names can be queried and completed.

Movement in the tree can also be accomplished via u (up), n (next), p (previous) and more. A question mark displays the commands.

Info can be exited with q.

Execution of commands by name

Any command can be executed using execute-extended-command (M-x). It is commonly used to execute commands that are not bound to a key sequence.

Example:

> command-apropos is bound to C-h a but it searches only commands. The more general command is apropos, which searches commands, functions, variables, and more.

To run apropos, use M-x. Type "apropos", then RET. apropos then prompts for a string to search for.

Execution of commands by name, continued

The key sequence C-x ESC ESC is bound to repeat-complex-command. It starts by displaying in the minibuffer the last "complex" command. The command is shown in the form of a Lisp expression.

The syntax of Lisp is among the simplest of all languages.

Rule 1: Just about everything in a body of Lisp code is a function call.

Rule 2: Function calls have this form:

(*function expr1 expr2 ... exprN*)

For example, following C-h f find-file RET, the key sequence C-x ESC ESC displays this:

```
Redo: (describe-function (quote find-file))
```

RET causes the describe-function call to be repeated.

M-p and M-n cycle through the commands.

Keyboard macros

It is sometimes useful to perform a series of editing operations several times.

An arbitrary series of keystrokes can be captured in a *keyboard macro* and repeatedly executed.

C-x (executes start-kbd-macro, which starts recording keystrokes.

C-x) is bound to stop-kbd-macro, which stops keystroke recording.

The recorded sequence can be played with C-x e (call-last-kbd-macro).

Example: Consider the steps to change lines from this:

```
/home/whm/x.java
/x/y.z
/a/bb/ccc/dddd/eeee
```

to this:

```
x.java    /home/whm
y.z   /x
eeee    /a/bb/ccc/dddd
```

The current keyboard macro can be assigned a name with name-last-kbd-macro. The macro can then be invoked with M-x using the assigned name.

C-u C-x (appends to a keyboard macro.

Sidebar: Start Emacs many times, or just once?

One way to use Emacs is to start up Emacs on a file, edit the file, save your changes, and exit Emacs. (Repeat as needed.)

A more common way to use Emacs is to start it in the morning, edit files all day, save buffers when necessary, and close it in the evening.

A simple way to do that is to have two ssh sessions. bash is run in one session; Emacs is run in the other. Alt+TAB or the mouse is used to switch between windows.

Sidebar, continued

bash's *job control* facility provides another way to shift between Emacs and the shell.

The key C-z is bound to suspend-emacs, which causes Emacs to immediately suspend execution and return control to the shell.

When you type C-z, the text displayed by Emacs will scroll up, a "Stopped" message will be printed, and the shell prompt appear:

```
[1]+  Stopped  emacs Hello.java
$
```

<u>The message "Stopped" is misleading: Emacs did not terminate; it is simply paused until the user resumes it.</u>

The user might then perform some number of shell commands. When the user is ready to continue editing, the fg (foreground) command is used:

```
$ fg 1  (Alternatives: %1, %emacs, and others)
```
[Emacs regains control of the terminal window]

The argument, `1`, is the job number, which was displayed in the "Stopped" message.

The jobs command shows what processes are suspended, if any:

```
$ jobs
[1]+  Stopped  emacs Hello.java
```

Sidebar, continued

A common error among novices is to accumulate several suspended Emacs jobs, each editing the same file:

```
$ jobs
[1]   Stopped  emacs fold.java
[2]-  Stopped  emacs fold.java
[3]+  Stopped  emacs fold.java
$
```

The common cause is that the user suspends Emacs and then, instead of resuming the suspended job, the user starts another Emacs job.

<u>The hazard of this situation is that if an older job is resumed, a save might overwrite a newer copy of the file with older buffer contents.</u> (Later creating a feeling that you're fixing a bug for the second time!)

This situation often leads to messages such as these:

```
fold.java has changed since visited or saved.  Save anyway?

fold.java locked by whm@lectura (pid 15883): (s, q, p, ?)?
```

If you see messages like these (and others), **<u>STOP!</u>** Use the `jobs` command to see what you've got running. Resume each suspended Emacs in turn and exit it. If you encounter one that has a modified (i.e., not saved) buffer, you can use `write-file` to save the buffer to an alternate file, for later examination to determine which version to keep.

Executive Summary: Don't run multiple Emacs jobs until you know what you're doing. Keep in mind that "Stopped" means "suspended".

Customization

When Emacs starts up, it looks for ~/.emacs. If found, it is assumed to contain Emacs Lisp code, which is then executed.

The behavior of Emacs can be changed and/or extended via code in ~/.emacs and additional Lisp files that .emacs causes to be loaded.

Emacs can be customized in a variety of ways. A simple customization is addition or alteration of key bindings.

The function global-set-key is used to bind a key sequence to a command.

Here is a call that binds C-t to the command other-window (normally on C-x o).

```
(global-set-key "\C-t"  'other-window)      ; 'other-window is an
                                            ; "atom"
```

eval-expression (M-:) is one way to immediately execute a Lisp expression. Type M-: and then (global-set-key "\C-t" 'other-window) at the "Eval:" prompt.

Executing the above expression binds C-t to other-window. The default binding of C-t (transpose-chars) is lost, but C-x o is still bound to other-window.

Customization, continued

Let's make the above binding, and three more, "permanent" by adding calls to ~/.emacs:

```
$ cat ~/.emacs
(global-set-key "\C-t" 'other-window)
(global-set-key "\M-a" 'beginning-of-buffer)
(global-set-key "\M-z" 'end-of-buffer)
(global-set-key [f2] 'find-file) ; binds F2 function key...
```

The next time Emacs is started, the bindings will be in effect.

Alternatively, load-file immediately executes an Emacs Lisp file.

The -q option of Emacs suppresses loading of ~/.emacs.

An obvious issue when changing bindings is that of losing the bindings that are displaced.

Important: Don't forget the backslash before a "C" or "M" in a binding.

Variables

The behavior of Emacs in many cases is controlled by the value of variables. The value of a variable can be changed with setq.

Examples:

```
(setq default-tab-width 4)
(setq stack-trace-on-error t)
(setq scroll-step 1)
(setq require-final-newline t)
```

Some variables, such as stack-trace-on-error, are essentially booleans. The value nil is considered to indicate "false". All other values are considered to indicate "true", but as a matter of style, t is used to indicate "true".

apropos-variable can be used to search variable names for a string, such as "tab".

describe-variable displays the documentation of a variable and also displays the current value of the variable.

Note: The function setq is not a command; it can't be executed via M-x. Use the set-variable command instead. (Or M-: (setq ...))

Functions

An Emacs Lisp function can be defined using defun.

Here is a function named 352-files that simply opens several files:

```
(defun 352-files ()
  (interactive)
  (find-file "~/352/problems.notes")
  (find-file "~/352/outline.notes")
  (find-file "~/352/emacs.notes")
  (find-file "~/352/notes")
  )
```

The call to interactive flags the function as a command, making it executable via M-x.

Functions, continued

Problem:

> The save-buffer command (C-x C-s) saves only the current buffer. It might be nice to have a way to save all modified buffers in a single operation.

Solution:

```
(defun save-all-buffers ()
  (interactive)
  (save-some-buffers t))

(global-set-key "\C-x\C-s" 'save-all-buffers)
```

save-all-buffers simply calls save-some-buffers with an argument indicating that all modified buffers are to be saved. It is then bound to C-x C-s, replacing save-buffer.

Functions, continued

Problem:

In some cases you might want to move the cursor to the start of a line but in others you might want the cursor positioned on the first non-blank character in the line.

One solution is to have two bindings. Here is another solution:

```
(defun first-non-whitespace ()
  (interactive)
  (beginning-of-line)
  (if (not (string= last-command 'first-non-whitespace))
    (skip-chars-forward " \t"))) ; blank and tab

(global-set-key "\C-a" 'first-non-whitespace)
```

The function is bound to `C-a`. If a single `C-a` is typed, the cursor is moved to the first non-whitespace character on the line. A successive `C-a` moves the cursor to the beginning of the line.

Bindings to lambda functions

Recall this example:

```
(defun save-all-buffers ()
  (interactive)
  (save-some-buffers t))

(global-set-key "\C-x\C-s" 'save-all-buffers)
```

Note that it is not sufficient to simply bind `save-some-buffers` to `C-x C-s` — an argument (`t`) must be passed. `save-all-buffers` simply "wraps" the call to `(save-some-buffers t)`.

If a function is used in only one place, a *lambda function* is a more concise alternative. Example:

```
(global-set-key "\C-x\C-s"
        '(lambda () (interactive) (save-some-buffers t)))
```

Another example:

```
(global-set-key "\C-t"
  '(lambda () (interactive) (beginning-of-line) (kill-line) (kill-line)))
```

Lambda functions are sometimes called *anonymous functions* because they have no name.

Modes

Emacs has *editing modes*—collections of customizations designed to facilitate editing various types of textual content.

There are *major modes* and *minor modes*.

A minor mode typically produces a slight variation in behavior.

Two examples of minor modes:

> Overwrite mode causes characters to replace existing characters. The command overwrite-mode (INS) toggles overwrite mode.
>
> Abbreviation mode (abbrev-mode) allows the user to define abbreviations that are expanded on-the-fly.

Major modes typically produce large variations in behavior. Emacs chooses among major modes based on the extension of the file being edited. Examples:

Extension(s)	*Mode*
java	java-mode
el	emacs-lisp-mode
c, h, y	c-mode
xml, dtd	sgml-mode

The current major mode (and minor mode(s), if any) are shown in the mode line.

describe-mode prints a description of the mode and shows mode-specific bindings.

Modes, continued

The variable auto-mode-alist, an *association list*, specifies the major mode associated with each extension. It is best viewed via M-x ielm.

Here is some abridged output:

```
*** Welcome to IELM ***  Type (describe-mode) for help.
ELISP> auto-mode-alist
(("\\.c\\'" . c-mode)
 ("\\.el\\'" . emacs-lisp-mode)
 ("\\.ad[abs]\\'" . ada-mode)
 ("\\.[12345678]\\'" . nroff-mode)
 ("\\.ms\\'" . nroff-mode)
("\\(/\\|\\`\\)\\.\\(bash_profile\\|z?login\\|bash_login\\|z?logout\\)\\'"
. sh-mode)
...
```

Modes, continued

Major modes for languages typically provide assistance with indentation, matching of paired tokens, comments, and navigation based on syntactic elements.

Some examples from C mode:

- As lines are typed, indentation is applied based on the current style and surrounding text. The TAB key does not simply insert a tab character. Instead it sets the indentation of the line.

- Parentheses and braces are matched as typed.

- M-a and M-e move the beginning or end of a statement, respectively.

- ESC C-h (c-mark-region) sets the region so that it surrounds the current function.

- indent-region sets the indentation on every line in the region.

- comment-region () comments out each line in the region using /* and */.

- C-c C-a toggles automatic insertion of newlines after certain syntactic elements.

Modes, continued

Modes typically involve ad-hoc syntactic analysis, which is notoriously prone to bugs.

Some modes are better than others. Example:

> Java mode is essentially a hacked-up C mode. `c-mark-region`, which marks a function in C, marks a Java class definition, not a method.

If a mode turns out to be a headache, one option is to switch to Fundamental mode with the `fundamental-mode` command. Fundamental mode is minimally "helpful".

All major modes can be disabled with this: `(setq auto-mode-alist nil)`

Running programs within Emacs

Emacs has excellent support for running programs and processing the output.

M-! (shell-command) prompts for a shell command line. The command is executed and the output is placed in the buffer *Shell Command Output*. (If the output is a single line it is displayed in the minibuffer, too.)

If argument is specified (C-u M-!), the output is placed in the current buffer.

M-| (shell-command-on-region) runs a (prompted for) command and supplies the contents of the region as standard input.

If an argument is specified for shell-command-on-region, the output of the command <u>replaces</u> the region.

Try this: Mark a region and then C-u M-| cat -n

The shell command runs an interactive shell in a buffer named *shell*.

Several bindings are established in shell-mode. For example, M-p steps through previous commands.

Backup files and auto save

Emacs creates a backup of an existing file the first time the file is saved in an Emacs session.

By default, the name of the backup file is formed by appending a tilde to the file name.

Example:

> The backup file for x.java would be x.java~, in the same directory as x.java.

The name of the backup file is generated by the function find-backup-file-name. A different naming scheme can be produced by replacing the function.

Here is a replacement that generates the name .ZBK.x.java.ZBK for x.java:

```
(defun find-backup-file-name (s)
   "Return a list containing the name of the backup name for s"
   (list (concat (file-name-directory s)
            ".ZBK."
            (file-name-nondirectory s)
            ".ZBK")))
```

Backup files and auto save, continued

A second form of protection is provided by auto-save. Auto saves are performed automatically based on keyboard activity (or inactivity). Additionally, an auto save is performed if Emacs is killed.

An auto save does not save into the original file. Instead, for a file named x, the auto save file is #x#.

An auto-save file is deleted whenever the file is saved.

Details of backup and auto-save operations are controlled by a variety of variables. (Use M-x apropos...)

Disabled commands

Some Emacs commands are initially disabled for a user.

One example is narrow-to-region (C-x n n)—a novice user might accidentally invoke it and seemingly lose much of a buffer.

If a disabled command is invoked, several choices are presented, including enabling it permanently. If that option is chosen then Emacs appends a line like this,

```
        (put 'narrow-to-region 'disabled nil)
```

to ~/.emacs.

Emacs and "real" windows

Emacs can be used very effectively in the terminal emulation environment of ssh clients but it also has excellent support for the X window system and Microsoft Windows.

If Emacs detects it is being run in a windowed environment it will open a window instead of running in the same window as the shell. (The -nw option suppresses this behavior.)

Emacs presents a "frame" with a menu and other GUI elements, but the frame contains an Emacs text window that operates the same as when in terminal mode.

Additional frames can be opened with Files | Make New Frame, or C-x 5 2, or with make-frame-command.

Various mouse and/or mouse and keyboard actions are bound to commands. Two examples:

> Sweeping over text with a mouse drag is the action drag-mouse-1 and is bound to mouse-set-region.
>
> C-down-mouse-1, a primary button click with the control pressed is bound to mouse-buffer-menu, which displays a menu of buffers.

Appendix: A simple ~/.emacs

```
(global-set-key "\C-t" 'other-window)
(global-set-key [f2] 'find-file) ; binds F2 function key...

(defun save-all-buffers ()
  (interactive)
  (save-some-buffers t))
(global-set-key "\C-x\C-s" 'save-all-buffers)

(defun first-non-whitespace ()
  (interactive)
  (beginning-of-line)
  (if (not (string= last-command 'first-non-whitespace))
    (skip-chars-forward " \t")))
(global-set-key "\C-a" 'first-non-whitespace)

(defun find-backup-file-name (s)
    "Return a list containing the name of the backup name for s"
    (interactive)
    (list (concat (file-name-directory s)
              ".ZBK."
              (file-name-nondirectory s)
              ".ZBK")))

(setq stack-trace-on-error t)
(setq require-final-newline t)
```