Ex Reference Manual
Version 3.7

William Joy

Mark Horton

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, Ca. 94720

ABSTRACT

Ex a line oriented text editor, which supports both command and display oriented editing. This reference manual describes the command oriented part of *ex;* the display editing features of *ex* are described in *An Introduction to Display Editing with Vi.* Other documents about the editor include the introduction *Edit: A tutorial*, the *Ex/edit Command Summary*, and a *Vi Quick Reference* card.

1. Starting ex

Each instance of the editor has a set of options, which can be set to tailor it to your liking. The command *edit* invokes a version of *ex* designed for more casual or beginning users by changing the default settings of some of these options. To simplify the description which follows we assume the default settings of the options.

When invoked, *ex* determines the terminal type from the TERM variable in the environment. It there is a TERMCAP variable in the environment, and the type of the terminal described there matches the TERM variable, then that description is used. Also if the TERMCAP variable contains a pathname (beginning with a /) then the editor will seek the description of the terminal in that file (rather than the default /etc/termcap). If there is a variable EXINIT in the environment, then the editor will execute the commands in that variable, otherwise if there is a file *.exrc* in your HOME directory *ex* reads commands from that file, simulating a *source* command. Option setting commands placed in EXINIT or *.exrc* will be executed before each editor session.

A command to enter *ex* has the following prototype:†

ex [–] [–**v**] [–**t** *tag*] [–**r**] [–**l**] [–**w***n*] [–**x**] [–**R**] [+*command*] name ...

The most common case edits a single file with no options, i.e.:

ex name

The – command line option option suppresses all interactive-user feedback and is useful in processing editor scripts in command files. The –**v** option is equivalent to using *vi* rather than *ex.* The –**t** option is equivalent to an initial *tag* command, editing the file containing the *tag* and positioning the editor at its definition. The –**r** option is used in recovering after an editor or system crash, retrieving the last saved version of the named file or, if no file is specified, typing a list of saved files. The –**l** option sets up for editing LISP, setting the *showmatch* and *lisp* options. The –**w** option sets the default window size to *n,* and is

The financial support of an IBM Graduate Fellowship and the National Science Foundation under grants MCS74-07644-A03 and MCS78-07291 is gratefully acknowledged.

† Brackets '[' ']' surround optional parameters here.

useful on dialups to start in small windows. The **−x** option causes *ex* to prompt for a *key*, which is used to encrypt and decrypt the contents of the file, which should already be encrypted using the same key, see *crypt*(1). The **−R** option sets the *readonly* option at the start. *Name* arguments indicate files to be edited. An argument of the form +*command* indicates that the editor should begin by executing the specified command. If *command* is omitted, then it defaults to ‘‘$’’, positioning the editor at the last line of the first file initially. Other useful commands here are scanning patterns of the form ‘‘/pat’’ or line numbers, e.g. ‘‘+100’’ starting at line 100.

2. File manipulation

2.1. Current file

Ex is normally editing the contents of a single file, whose name is recorded in the *current* file name. *Ex* performs all editing actions in a buffer (actually a temporary file) into which the text of the file is initially read. Changes made to the buffer have no effect on the file being edited unless and until the buffer contents are written out to the file with a *write* command. After the buffer contents are written, the previous contents of the written file are no longer accessible. When a file is edited, its name becomes the current file name, and its contents are read into the buffer.

The current file is almost always considered to be *edited.* This means that the contents of the buffer are logically connected with the current file name, so that writing the current buffer contents onto that file, even if it exists, is a reasonable action. If the current file is not *edited* then *ex* will not normally write on it if it already exists.*

2.2. Alternate file

Each time a new value is given to the current file name, the previous current file name is saved as the *alternate* file name. Similarly if a file is mentioned but does not become the current file, it is saved as the alternate file name.

2.3. Filename expansion

Filenames within the editor may be specified using the normal shell expansion conventions. In addition, the character ‘%’ in filenames is replaced by the *current* file name and the character ‘#’ by the *alternate* file name.†

2.4. Multiple files and named buffers

If more than one file is given on the command line, then the first file is edited as described above. The remaining arguments are placed with the first file in the *argument list.* The current argument list may be displayed with the *args* command. The next file in the argument list may be edited with the *next* command. The argument list may also be respecified by specifying a list of names to the *next* command. These names are expanded, the resulting list of names becomes the new argument list, and *ex* edits the first file on the list.

For saving blocks of text while editing, and especially when editing more than one file, *ex* has a group of named buffers. These are similar to the normal buffer, except that only a limited number of operations are available on them. The buffers have names *a* through *z.*‡

2.5. Read only

It is possible to use *ex* in *read only* mode to look at files that you have no intention of modifying. This mode protects you from accidently overwriting the file. Read only mode is on when the *readonly*

* The *file* command will say ‘‘[Not edited]’’ if the current file is not considered edited.

† This makes it easy to deal alternately with two files and eliminates the need for retyping the name supplied on an *edit* command after a *No write since last change* diagnostic is received.

‡ It is also possible to refer to *A* through *Z;* the upper case buffers are the same as the lower but commands append to named buffers rather than replacing if upper case names are used.

option is set. It can be turned on with the **−R** command line option, by the *view* command line invocation, or by setting the *readonly* option. It can be cleared by setting *noreadonly*. It is possible to write, even while in read only mode, by indicating that you really know what you are doing. You can write to a different file, or can use the ! form of write, even while in read only mode.

3. Exceptional Conditions

3.1. Errors and interrupts

When errors occur *ex* (optionally) rings the terminal bell and, in any case, prints an error diagnostic. If the primary input is from a file, editor processing will terminate. If an interrupt signal is received, *ex* prints ''Interrupt'' and returns to its command level. If the primary input is a file, then *ex* will exit when this occurs.

3.2. Recovering from hangups and crashes

If a hangup signal is received and the buffer has been modified since it was last written out, or if the system crashes, either the editor (in the first case) or the system (after it reboots in the second) will attempt to preserve the buffer. The next time you log in you should be able to recover the work you were doing, losing at most a few lines of changes from the last point before the hangup or editor crash. To recover a file you can use the **−r** option. If you were editing the file *resume,* then you should change to the directory where you were when the crash occurred, giving the command

ex −r *resume*

After checking that the retrieved file is indeed ok, you can *write* it over the previous contents of that file.

You will normally get mail from the system telling you when a file has been saved after a crash. The command

ex −r

will print a list of the files which have been saved for you. (In the case of a hangup, the file will not appear in the list, although it can be recovered.)

4. Editing modes

Ex has five distinct modes. The primary mode is *command* mode. Commands are entered in command mode when a ':' prompt is present, and are executed each time a complete line is sent. In *text input* mode *ex* gathers input lines and places them in the file. The *append, insert,* and *change* commands use text input mode. No prompt is printed when you are in text input mode. This mode is left by typing a '.' alone at the beginning of a line, and *command* mode resumes.

The last three modes are *open* and *visual* modes, entered by the commands of the same name, and, within open and visual modes *text insertion* mode. *Open* and *visual* modes allow local editing operations to be performed on the text in the file. The *open* command displays one line at a time on any terminal while *visual* works on CRT terminals with random positioning cursors, using the screen as a (single) window for file editing changes. These modes are described (only) in *An Introduction to Display Editing with Vi.*

5. Command structure

Most command names are English words, and initial prefixes of the words are acceptable abbreviations. The ambiguity of abbreviations is resolved in favor of the more commonly used commands.*

---

* As an example, the command *substitute* can be abbreviated 's' while the shortest available abbreviation for the *set* command is 'se'.

5.1. Command parameters

Most commands accept prefix addresses specifying the lines in the file upon which they are to have effect. The forms of these addresses will be discussed below. A number of commands also may take a trailing *count* specifying the number of lines to be involved in the command.† Thus the command ‘‘10p’’ will print the tenth line in the buffer while ‘‘delete 5’’ will delete five lines from the buffer, starting with the current line.

Some commands take other information or parameters, this information always being given after the command name.‡

5.2. Command variants

A number of commands have two distinct variants. The variant form of the command is invoked by placing an ‘!’ immediately after the command name. Some of the default variants may be controlled by options; in this case, the ‘!’ serves to toggle the default.

5.3. Flags after commands

The characters ‘#’, ‘p’ and ‘l’ may be placed after many commands.** In this case, the command abbreviated by these characters is executed after the command completes. Since *ex* normally prints the new current line after each change, ‘p’ is rarely necessary. Any number of ‘+’ or ‘−’ characters may also be given with these flags. If they appear, the specified offset is applied to the current line value before the printing command is executed.

5.4. Comments

It is possible to give editor commands which are ignored. This is useful when making complex editor scripts for which comments are desired. The comment character is the double quote: ". Any command line beginning with " is ignored. Comments beginning with " may also be placed at the ends of commands, except in cases where they could be confused as part of text (shell escapes and the substitute and map commands).

5.5. Multiple commands per line

More than one command may be placed on a line by separating each pair of commands by a ‘|’ character. However the *global* commands, comments, and the shell escape ‘!’ must be the last command on a line, as they are not terminated by a ‘|’.

5.6. Reporting large changes

Most commands which change the contents of the editor buffer give feedback if the scope of the change exceeds a threshold given by the *report* option. This feedback helps to detect undesirably large changes so that they may be quickly and easily reversed with an *undo.* After commands with more global effect such as *global* or *visual,* you will be informed if the net change in the number of lines in the buffer during this command exceeds this threshold.

6. Command addressing

6.1. Addressing primitives

. The current line. Most commands leave the current line as the last line which they affect. The default address for most commands is the current line, thus ‘**.**’ is rarely used alone as an address.

---

† Counts are rounded down if necessary.

‡ Examples would be option names in a *set* command i.e. ‘‘set number’’, a file name in an *edit* command, a regular expression in a *substitute* command, or a target address for a *copy* command, i.e. ‘‘1,5 copy 25’’.

** A ‘p’ or ‘l’ must be preceded by a blank or tab except in the single special case ‘dp’.

n	The *n*th line in the editor's buffer, lines being numbered sequentially from 1.
$	The last line in the buffer.
%	An abbreviation for ''1,$'', the entire buffer.
+n −n	An offset relative to the current buffer line.†
/*pat*/ **?***pat***?**	Scan forward and backward respectively for a line containing *pat*, a regular expression (as defined below). The scans normally wrap around the end of the buffer. If all that is desired is to print the next line containing *pat*, then the trailing **/** or **?** may be omitted. If *pat* is omitted or explicitly empty, then the last regular expression specified is located.‡
´´ ´*x*	Before each non-relative motion of the current line '**.**', the previous current line is marked with a tag, subsequently referred to as '´´'. This makes it easy to refer or return to this previous context. Marks may also be established by the *mark* command, using single lower case letters *x* and the marked lines referred to as '´*x*'.

6.2. Combining addressing primitives

Addresses to commands consist of a series of addressing primitives, separated by ',' or ';'. Such address lists are evaluated left-to-right. When addresses are separated by ';' the current line '**.**' is set to the value of the previous addressing expression before the next address is interpreted. If more addresses are given than the command requires, then all but the last one or two are ignored. If the command takes two addresses, the first addressed line must precede the second in the buffer.†

7. Command descriptions

The following form is a prototype for all *ex* commands:

address **command** *! parameters count flags*

All parts are optional; the degenerate case is the empty command which prints the next line in the file. For sanity with use from within *visual* mode, *ex* ignores a '':'' preceding any command.

In the following command descriptions, the default addresses are shown in parentheses, which are *not,* however, part of the command.

abbreviate *word rhs* abbr: **ab**

Add the named abbreviation to the current list. When in input mode in visual, if *word* is typed as a complete word, it will be changed to *rhs*.

(**.**) **append** abbr: **a**
text
.

Reads the input text and places it after the specified line. After the command, '**.**' addresses the last line input or the specified line if no lines were input. If address '0' is given, text is placed at the beginning of the buffer.

a!
text
.

The variant flag to *append* toggles the setting for the *autoindent* option during the input of *text.*

---

† The forms '.+3' '+3' and '+++' are all equivalent; if the current line is line 100 they all address line 103.
‡ The forms \/ and **?** scan using the last regular expression used in a scan; after a substitute **//** and **??** would scan using the substitute's regular expression.
† Null address specifications are permitted in a list of addresses, the default in this case is the current line '.'; thus ',100' is equivalent to '**.**,100'. It is an error to give a prefix address to a command which expects none.

args

The members of the argument list are printed, with the current argument delimited by '[' and ']'.

(**.** , **.**) **change** *count* abbr: **c**
text
.

Replaces the specified lines with the input *text*. The current line becomes the last line input; if no lines were input it is left as for a *delete*.

c!
text
.

The variant toggles *autoindent* during the *change.*

(**.** , **.**) **copy** *addr flags* abbr: **co**

A *copy* of the specified lines is placed after *addr,* which may be '0'. The current line '**.**' addresses the last line of the copy. The command *t* is a synonym for *copy.*

(**.** , **.**) **delete** *buffer count flags* abbr: **d**

Removes the specified lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line. If a named *buffer* is specified by giving a letter, then the specified lines are saved in that buffer, or appended to it if an upper case letter is used.

edit *file* abbr: **e**
ex *file*

Used to begin an editing session on a new file. The editor first checks to see if the buffer has been modified since the last *write* command was issued. If it has been, a warning is issued and the command is aborted. The command otherwise deletes the entire contents of the editor buffer, makes the named file the current file and prints the new filename. After insuring that this file is sensible† the editor reads the file into its buffer.

If the read of the file completes without error, the number of lines and characters read is typed. If there were any non-ASCII characters in the file they are stripped of their non-ASCII high bits, and any null characters in the file are discarded. If none of these errors occurred, the file is considered *edited.* If the last line of the input file is missing the trailing newline character, it will be supplied and a complaint will be issued. This command leaves the current line '**.**' at the last line read.‡

e! *file*

The variant form suppresses the complaint about modifications having been made and not written from the editor buffer, thus discarding all changes which have been made before editing the new file.

e +*n file*

Causes the editor to begin at line *n* rather than at the last line; *n* may also be an editor command containing no spaces, e.g.: ''+/pat''.

† I.e., that it is not a binary file such as a directory, a block or character special file other than */dev/tty,* a terminal, or a binary or executable file (as indicated by the first word).
‡ If executed from within *open* or *visual,* the current line is initially the first line of the file.

file abbr: **f**

Prints the current file name, whether it has been '[Modified]' since the last *write* command, whether it is *read only*, the current line, the number of lines in the buffer, and the percentage of the way through the buffer of the current line.*

file *file*

The current file name is changed to *file* which is considered '[Not edited]'.

(1 , $) **global** /*pat* / *cmds* abbr: **g**

First marks each line among those specified which matches the given regular expression. Then the given command list is executed with '**.**' initially set to each marked line.

The command list consists of the remaining commands on the current input line and may continue to multiple lines by ending all but the last such line with a '\'. If *cmds* (and possibly the trailing **/** delimiter) is omitted, each line matching *pat* is printed. *Append, insert,* and *change* commands and associated input are permitted; the '**.**' terminating input may be omitted if it would be on the last line of the command list. *Open* and *visual* commands are permitted in the command list and take input from the terminal.

The *global* command itself may not appear in *cmds.* The *undo* command is also not permitted there, as *undo* instead can be used to reverse the entire *global* command. The options *autoprint* and *autoindent* are inhibited during a *global,* (and possibly the trailing **/** delimiter) and the value of the *report* option is temporarily infinite, in deference to a *report* for the entire global. Finally, the context mark '´´' is set to the value of '.' before the global command begins and is not changed during a global command, except perhaps by an *open* or *visual* within the *global.*

g! /*pat*/ *cmds* abbr: **v**

The variant form of *global* runs *cmds* at each line not matching *pat*.

(**.**) **insert** abbr: **i**
text
.

Places the given text before the specified line. The current line is left at the last line input; if there were none input it is left at the line before the addressed line. This command differs from *append* only in the placement of text.

i!
text
.

The variant toggles *autoindent* during the *insert.*

(**.** , **.**+1) **join** *count flags* abbr: **j**

Places the text from a specified range of lines together on one line. White space is adjusted at each junction to provide at least one blank character, two if there was a '**.**' at the end of the line, or none if the first following character is a ')'. If there is already white space at the end of the line, then the white space at the start of the next line will be discarded.

* In the rare case that the current file is '[Not edited]' this is noted also; in this case you have to use the form **w!** to write to the file, since the editor is not sure that a **write** will not destroy a file unrelated to the current contents of the buffer.

j!

The variant causes a simpler *join* with no white space processing; the characters in the lines are simply concatenated.

(**.**) **k** *x*

The *k* command is a synonym for *mark.* It does not require a blank or tab before the following letter.

(**.** , **.**) **list** *count flags*

Prints the specified lines in a more unambiguous way: tabs are printed as ‘^I’ and the end of each line is marked with a trailing ‘$’. The current line is left at the last line printed.

map *lhs rhs*

The *map* command is used to define macros for use in *visual* mode. *Lhs* should be a single character, or the sequence ‘‘#n’’, for n a digit, referring to function key *n*. When this character or function key is typed in *visual* mode, it will be as though the corresponding *rhs* had been typed. On terminals without function keys, you can type ‘‘#n’’. See section 6.9 of the ‘‘Introduction to Display Editing with Vi’’ for more details.

(**.**) **mark** *x*

Gives the specified line mark *x,* a single lower case letter. The *x* must be preceded by a blank or a tab. The addressing form ‘´x’ then addresses this line. The current line is not affected by this command.

(**.** , **.**) **move** *addr* abbr: **m**

The *move* command repositions the specified lines to be after *addr*. The first of the moved lines becomes the current line.

next abbr: **n**

The next file from the command line argument list is edited.

n!

The variant suppresses warnings about the modifications to the buffer not having been written out, discarding (irretrievably) any changes which may have been made.

n *filelist*
n +*command filelist*

The specified *filelist* is expanded and the resulting list replaces the current argument list; the first file in the new list is then edited. If *command* is given (it must contain no spaces), then it is executed after editing the first such file.

(**.** , **.**) **number** *count flags* abbr: **#** or **nu**

Prints each specified line preceded by its buffer line number. The current line is left at the last line printed.

(**.**) **open** *flags* abbr: **o**
(**.**) **open** /*pat*/ *flags*

Enters intraline editing *open* mode at each addressed line. If *pat* is given, then the cursor will be placed initially at the beginning of the string matched by the pattern. To exit this mode use Q. See *An Introduction to Display Editing with Vi* for more details.

preserve

The current editor buffer is saved as though the system had just crashed. This command is for use only in emergencies when a *write* command has resulted in an error and you don't know how to save your work. After a *preserve* you should seek help.

(**.** , **.**) **print** *count* abbr: **p** or **P**

Prints the specified lines with non-printing characters printed as control characters '^*x*'; delete (octal 177) is represented as '^?'. The current line is left at the last line printed.

(**.**) **put** *buffer* abbr: **pu**

Puts back previously *deleted* or *yanked* lines. Normally used with *delete* to effect movement of lines, or with *yank* to effect duplication of lines. If no *buffer* is specified, then the last *deleted* or *yanked* text is restored.* By using a named buffer, text may be restored that was saved there at any previous time.

quit abbr: **q**

Causes *ex* to terminate. No automatic write of the editor buffer to a file is performed. However, *ex* issues a warning message if the file has changed since the last *write* command was issued, and does not *quit.*† Normally, you will wish to save your changes, and you should give a *write* command; if you wish to discard them, use the **q!** command variant.

q!

Quits from the editor, discarding changes to the buffer without complaint.

(**.**) **read** *file* abbr: **r**

Places a copy of the text of the given file in the editing buffer after the specified line. If no *file* is given the current file name is used. The current file name is not changed unless there is none in which case *file* becomes the current name. The sensibility restrictions for the *edit* command apply here also. If the file buffer is empty and there is no current name then *ex* treats this as an *edit* command.

Address '0' is legal for this command and causes the file to be read at the beginning of the buffer. Statistics are given as for the *edit* command when the *read* successfully terminates. After a *read* the current line is the last line read.‡

(**.**) **read !***command*

Reads the output of the command *command* into the buffer after the specified line. This is not a variant form of the command, rather a read specifying a *command* rather than a *filename;* a blank or tab before the **!** is mandatory.

recover *file*

Recovers *file* from the system save area. Used after a accidental hangup of the phone** or a system crash** or *preserve* command. Except when you use *preserve* you will be notified by mail when a file is saved.

---

* But no modifying commands may intervene between the *delete* or *yank* and the *put,* nor may lines be moved between files without using a named buffer.

† *Ex* will also issue a diagnostic if there are more files in the argument list.

‡ Within *open* and *visual* the current line is set to the first line read rather than the last.

** The system saves a copy of the file you were editing only if you have made changes to the file.

rewind abbr: **rew**

The argument list is rewound, and the first file in the list is edited.

rew!

Rewinds the argument list discarding any changes made to the current buffer.

set *parameter*

With no arguments, prints those options whose values have been changed from their defaults; with parameter *all* it prints all of the option values.

Giving an option name followed by a '?' causes the current value of that option to be printed. The '?' is unnecessary unless the option is Boolean valued. Boolean options are given values either by the form 'set *option*' to turn them on or 'set no*option*' to turn them off; string and numeric options are assigned via the form 'set *option*=value'.

More than one parameter may be given to *set* ; they are interpreted left-to-right.

shell abbr: **sh**

A new shell is created. When it terminates, editing resumes.

source *file* abbr: **so**

Reads and executes commands from the specified file. *Source* commands may be nested.

(**.** , **.**) **substitute** /*pat* /*repl* / *options count flags* abbr: **s**

On each specified line, the first instance of pattern *pat* is replaced by replacement pattern *repl.* If the *global* indicator option character 'g' appears, then all instances are substituted; if the *confirm* indication character 'c' appears, then before each substitution the line to be substituted is typed with the string to be substituted marked with '↑' characters. By typing an 'y' one can cause the substitution to be performed, any other input causes no change to take place. After a *substitute* the current line is the last line substituted.

Lines may be split by substituting new-line characters into them. The newline in *repl* must be escaped by preceding it with a '\'. Other metacharacters available in *pat* and *repl* are described below.

stop

Suspends the editor, returning control to the top level shell. If *autowrite* is set and there are unsaved changes, a write is done first unless the form **stop**! is used. This commands is only available where supported by the teletype driver and operating system.

(**.** , **.**) **substitute** *options count flags* abbr: **s**

If *pat* and *repl* are omitted, then the last substitution is repeated. This is a synonym for the **&** command.

(**.** , **.**) **t** *addr flags*

The *t* command is a synonym for *copy* .

ta *tag*

The focus of editing switches to the location of *tag,* switching to a different line in the current file where it is defined, or if necessary to another file.‡

‡ If you have modified the current file before giving a *tag* command, you must write it out; giving another *tag* command, specifying no *tag* will reuse the previous tag.

The tags file is normally created by a program such as *ctags,* and consists of a number of lines with three fields separated by blanks or tabs. The first field gives the name of the tag, the second the name of the file where the tag resides, and the third gives an addressing form which can be used by the editor to find the tag; this field is usually a contextual scan using '/*pat*/' to be immune to minor changes in the file. Such scans are always performed as if *nomagic* was set.

The tag names in the tags file must be sorted alphabetically.

unabbreviate *word* abbr: **una**

Delete *word* from the list of abbreviations.

undo abbr: **u**

Reverses the changes made in the buffer by the last buffer editing command. Note that *global* commands are considered a single command for the purpose of *undo* (as are *open* and *visual.)* Also, the commands *write* and *edit* which interact with the file system cannot be undone. *Undo* is its own inverse.

Undo always marks the previous value of the current line '**.**' as '´´'. After an *undo* the current line is the first line restored or the line before the first line deleted if no lines were restored. For commands with more global effect such as *global* and *visual* the current line regains it's pre-command value after an *undo.*

unmap *lhs*

The macro expansion associated by *map* for *lhs* is removed.

(1 , $) **v** /*pat* / *cmds*

A synonym for the *global* command variant **g!**, running the specified *cmds* on each line which does not match *pat.*

version abbr: **ve**

Prints the current version number of the editor as well as the date the editor was last changed.

(**.**) **visual** *type count flags* abbr: **vi**

Enters visual mode at the specified line. *Type* is optional and may be '–' , '↑' or '**.**' as in the *z* command to specify the placement of the specified line on the screen. By default, if *type* is omitted, the specified line is placed as the first on the screen. A *count* specifies an initial window size; the default is the value of the option *window.* See the document *An Introduction to Display Editing with Vi* for more details. To exit this mode, type Q.

visual file
visual +*n* file

From visual mode, this command is the same as edit.

(1 , $) **write** *file* abbr: **w**

Writes changes made back to *file*, printing the number of lines and characters written. Normally *file* is omitted and the text goes back where it came from. If a *file* is specified, then text will be written to that file.* If the file does not exist it is created. The current file name is changed only if there is no current file name; the current line is never changed.

If an error occurs while writing the current and *edited* file, the editor considers that there has been ''No write since last change'' even if the buffer had not previously been modified.

---

* The editor writes to a file only if it is the current file and is *edited* , if the file does not exist, or if the file is actually a teletype, */dev/tty,* */dev/null.* Otherwise, you must give the variant form **w!** to force the write.

(1 , $) **write>>** *file* abbr: **w>>**

Writes the buffer contents at the end of an existing file.

w! *name*

Overrides the checking of the normal *write* command, and will write to any file which the system permits.

(1 , $) **w !***command*

Writes the specified lines into *command.* Note the difference between **w!** which overrides checks and **w !** which writes to a command.

wq *name*

Like a *write* and then a *quit* command.

wq! *name*

The variant overrides checking on the sensibility of the *write* command, as **w!** does.

xit *name*

If any changes have been made and not written, writes the buffer out. Then, in any case, quits.

(**. , .**) **yank** *buffer count* abbr: **ya**

Places the specified lines in the named *buffer,* for later retrieval via *put.* If no buffer name is specified, the lines go to a more volatile place; see the *put* command description.

(**.+1**) **z** *count*

Print the next *count* lines, default *window*.

(**.**) **z** *type count*

Prints a window of text with the specified line at the top. If *type* is '−' the line is placed at the bottom; a '**.**' causes the line to be placed in the center.* A count gives the number of lines to be displayed rather than double the number specified by the *scroll* option. On a CRT the screen is cleared before display begins unless a count which is less than the screen size is given. The current line is left at the last line printed.

! *command*

The remainder of the line after the '!' character is sent to a shell to be executed. Within the text of *command* the characters '%' and '#' are expanded as in filenames and the character '!' is replaced with the text of the previous command. Thus, in particular, '!!' repeats the last such shell escape. If any such expansion is performed, the expanded line will be echoed. The current line is unchanged by this command.

If there has been ''[No write]'' of the buffer contents since the last change to the editing buffer, then a diagnostic will be printed before the command is executed as a warning. A single '!' is printed when the command completes.

* Forms 'z=' and 'z↑' also exist; 'z=' places the current line in the center, surrounds it with lines of '–' characters and leaves the current line at this line. The form 'z↑' prints the window before 'z–' would. The characters '+', '↑' and '–' may be repeated for cumulative effect. On some v2 editors, no *type* may be given.

(*addr* , *addr*) **!** *command*

Takes the specified address range and supplies it as standard input to *command;* the resulting output then replaces the input lines.

($) =

Prints the line number of the addressed line. The current line is unchanged.

(**.** , **.**) > *count flags*
(**.** , **.**) < *count flags*

Perform intelligent shifting on the specified lines; **<** shifts left and **>** shift right. The quantity of shift is determined by the *shiftwidth* option and the repetition of the specification character. Only white space (blanks and tabs) is shifted; no non-white characters are discarded in a left-shift. The current line becomes the last line which changed due to the shifting.

ˆD

An end-of-file from a terminal input scrolls through the file. The *scroll* option specifies the size of the scroll, normally a half screen of text.

(**.**+1 , **.**+1)
(**.**+1 , **.**+1) |

An address alone causes the addressed lines to be printed. A blank line prints the next line in the file.

(**.** , **.**) **&** *options count flags*

Repeats the previous *substitute* command.

(**.** , **.**) **˜** *options count flags*

Replaces the previous regular expression with the previous replacement pattern from a substitution.

8. Regular expressions and substitute replacement patterns

8.1. Regular expressions

A regular expression specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. *Ex* remembers two previous regular expressions: the previous regular expression used in a *substitute* command and the previous regular expression used elsewhere (referred to as the previous *scanning* regular expression.) The previous regular expression can always be referred to by a null *re*, e.g. '//' or '??'.

8.2. Magic and nomagic

The regular expressions allowed by *ex* are constructed in one of two ways depending on the setting of the *magic* option. The *ex* and *vi* default setting of *magic* gives quick access to a powerful set of regular expression metacharacters. The disadvantage of *magic* is that the user must remember that these metacharacters are *magic* and precede them with the character '\' to use them as ''ordinary'' characters. With *nomagic,* the default for *edit,* regular expressions are much simpler, there being only two metacharacters. The power of the other metacharacters is still available by preceding the (now) ordinary character with a '\'. Note that '\' is thus always a metacharacter.

The remainder of the discussion of regular expressions assumes that that the setting of this option is *magic.*†

† To discern what is true with *nomagic* it suffices to remember that the only special characters in this case will be '↑' at the beginning of a regular expression, '$' at the end of a regular expression, and '\'. With *nomagic* the characters '˜' and '&' also lose their special meanings related to the replacement pattern of a substitute.

8.3. Basic regular expression summary

The following basic constructs are used to construct *magic* mode regular expressions.

char An ordinary character matches itself. The characters ‘↑’ at the beginning of a line, ‘$’ at the end of line, ‘*’ as any character other than the first, ‘.’, ‘\’, ‘[’, and ‘~’ are not ordinary characters and must be escaped (preceded) by ‘\’ to be treated as such.

↑ At the beginning of a pattern forces the match to succeed only at the beginning of a line.

$ At the end of a regular expression forces the match to succeed only at the end of the line.

. Matches any single character except the new-line character.

\< Forces the match to occur only at the beginning of a ‘‘variable’’ or ‘‘word’’; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.

\> Similar to ‘\<’, but matching the end of a ‘‘variable’’ or ‘‘word’’, i.e. either the end of the line or before character which is neither a letter, nor a digit, nor the underline character.

[*string*] Matches any (single) character in the class defined by *string.* Most characters in *string* define themselves. A pair of characters separated by ‘–’ in *string* defines the set of characters collating between the specified lower and upper bounds, thus ‘[a–z]’ as a regular expression matches any (single) lower-case letter. If the first character of *string* is an ‘↑’ then the construct matches those characters which it otherwise would not; thus ‘[↑a–z]’ matches anything but a lower-case letter (and of course a newline). To place any of the characters ‘↑’, ‘[’, or ‘–’ in *string* you must escape them with a preceding ‘\’.

8.4. Combining regular expression primitives

The concatenation of two regular expressions matches the leftmost and then longest string which can be divided with the first piece matching the first regular expression and the second piece matching the second. Any of the (single character matching) regular expressions mentioned above may be followed by the character ‘*’ to form a regular expression which matches any number of adjacent occurrences (including 0) of characters matched by the regular expression it follows.

The character ‘~’ may be used in a regular expression, and matches the text which defined the replacement part of the last *substitute* command. A regular expression may be enclosed between the sequences ‘\(’ and ‘\)’ with side effects in the *substitute* replacement patterns.

8.5. Substitute replacement patterns

The basic metacharacters for the replacement pattern are ‘&’ and ‘~’; these are given as ‘\&’ and ‘\~’ when *nomagic* is set. Each instance of ‘&’ is replaced by the characters which the regular expression matched. The metacharacter ‘~’ stands, in the replacement pattern, for the defining text of the previous replacement pattern.

Other metasequences possible in the replacement pattern are always introduced by the escaping character ‘\’. The sequence ‘*n*’ is replaced by the text matched by the *n*-th regular subexpression enclosed between ‘\(’ and ‘\)’.† The sequences ‘\u’ and ‘\l’ cause the immediately following character in the replacement to be converted to upper- or lower-case respectively if this character is a letter. The sequences ‘\U’ and ‘\L’ turn such conversion on, either until ‘\E’ or ‘\e’ is encountered, or until the end of the replacement pattern.

† When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of ‘\(’ starting from the left.

9. Option descriptions

autoindent, **ai** default: noai

Can be used to ease the preparation of structured program text. At the beginning of each *append*, *change* or *insert* command or when a new line is *opened* or created by an *append*, *change*, *insert*, or *substitute* operation within *open* or *visual* mode, *ex* looks at the line being appended after, the first line changed or the line inserted before and calculates the amount of white space at the start of the line. It then aligns the cursor at the level of indentation so determined.

If the user then types lines of text in, they will continue to be justified at the displayed indenting level. If more white space is typed at the beginning of a line, the following line will start aligned with the first non-white character of the previous line. To back the cursor up to the preceding tab stop one can hit **ˆD**. The tab stops going backwards are defined at multiples of the *shiftwidth* option. You *cannot* backspace over the indent, except by sending an end-of-file with a **ˆD**.

Specially processed in this mode is a line with no characters added to it, which turns into a completely blank line (the white space provided for the *autoindent* is discarded.) Also specially processed in this mode are lines beginning with an '↑' and immediately followed by a **ˆD**. This causes the input to be repositioned at the beginning of the line, but retaining the previous indent for the next line. Similarly, a '0' followed by a **ˆD** repositions at the beginning but without retaining the previous indent.

Autoindent doesn't happen in *global* commands or when the input is not a terminal.

autoprint, **ap** default: ap

Causes the current line to be printed after each *delete*, *copy*, *join*, *move*, *substitute*, *t*, *undo* or shift command. This has the same effect as supplying a trailing 'p' to each such command. *Autoprint* is suppressed in globals, and only applies to the last of many commands on a line.

autowrite, **aw** default: noaw

Causes the contents of the buffer to be written to the current file if you have modified it and give a *next, rewind, stop, tag,* or *!* command, or a ˆ↑ (switch files) or **ˆ]** (tag goto) command in *visual.* Note, that the *edit* and *ex* commands do **not** autowrite. In each case, there is an equivalent way of switching when autowrite is set to avoid the *autowrite* (*edit* for *next*, *rewind!* for .I rewind , *stop!* for *stop*, *tag!* for *tag*, *shell* for *!*, and **:e #** and a **:ta!** command from within *visual).*

beautify, **bf** default: nobeautify

Causes all control characters except tab, newline and form-feed to be discarded from the input. A complaint is registered the first time a backspace character is discarded. *Beautify* does not apply to command input.

directory, **dir** default: dir=/tmp

Specifies the directory in which *ex* places its buffer file. If this directory in not writable, then the editor will exit abruptly when it fails to be able to create its buffer there.

edcompatible default: noedcompatible

Causes the presence of absence of **g** and **c** suffixes on substitute commands to be remembered, and to be toggled by repeating the suffices. The suffix **r** makes the substitution be as in the ˜ command, instead of like *&*.

errorbells, **eb** default: noeb

Error messages are preceded by a bell.* If possible the editor always places the error message in a

* Bell ringing in *open* and *visual* on errors is not suppressed by setting *noeb.*

standout mode of the terminal (such as inverse video) instead of ringing the bell.

hardtabs, **ht** default: ht=8

Gives the boundaries on which terminal hardware tabs are set (or on which the system expands tabs).

ignorecase, **ic** default: noic

All upper case characters in the text are mapped to lower case in regular expression matching. In addition, all upper case characters in regular expressions are mapped to lower case except in character class specifications.

lisp default: nolisp

Autoindent indents appropriately for *lisp* code, and the **( ) { } [[** and **]]** commands in *open* and *visual* are modified to have meaning for *lisp*.

list default: nolist

All printed lines will be displayed (more) unambiguously, showing tabs and end-of-lines as in the *list* command.

magic default: magic for *ex* and *vi*†

If *nomagic* is set, the number of regular expression metacharacters is greatly reduced, with only '↑' and '$' having special effects. In addition the metacharacters '~' and '&' of the replacement pattern are treated as normal characters. All the normal metacharacters may be made *magic* when *nomagic* is set by preceding them with a '\'.

mesg default: mesg

Causes write permission to be turned off to the terminal while you are in visual mode, if *nomesg* is set.

modeline default: nomodeline

If *modeline* is set, then the first 5 lines and the last five lines of the file will be checked for ex command lines and the comands issued. To be recognized as a command line, the line must have the string **ex:** or **vi:** preceeded by a tab or a space. This string may be anywhere in the line and anything after the *:* is interpeted as editor commands. This option defaults to off because of unexpected behavior when editting files such as */etc/passwd.*

number, nu default: nonumber

Causes all output lines to be printed with their line numbers. In addition each input line will be prompted for by supplying the line number it will have.

open default: open

If *noopen*, the commands *open* and *visual* are not permitted. This is set for *edit* to prevent confusion resulting from accidental entry to open or visual mode.

optimize, opt default: optimize

Throughput of text is expedited by setting the terminal to not do automatic carriage returns when printing more than one (logical) line of output, greatly speeding output on terminals without addressable cursors when text with leading white space is printed.

† *Nomagic* for *edit*.

paragraphs, para default: para=IPLPPPQPP LIbp

Specifies the paragraphs for the **{** and **}** operations in *open* and *visual.* The pairs of characters in the option's value are the names of the macros which start paragraphs.

prompt default: prompt

Command mode input is prompted for with a ':'.

redraw default: noredraw

The editor simulates (using great amounts of output), an intelligent terminal on a dumb terminal (e.g. during insertions in *visual* the characters to the right of the cursor position are refreshed as each input character is typed.) Useful only at very high speed.

remap default: remap

If on, macros are repeatedly tried until they are unchanged. For example, if **o** is mapped to **O**, and **O** is mapped to **I**, then if *remap* is set, **o** will map to **I**, but if *noremap* is set, it will map to **O**.

report default: report=5†

Specifies a threshold for feedback from commands. Any command which modifies more than the specified number of lines will provide feedback as to the scope of its changes. For commands such as *global*, *open*, *undo*, and *visual* which have potentially more far reaching scope, the net change in the number of lines in the buffer is presented at the end of the command, subject to this same threshold. Thus notification is suppressed during a *global* command on the individual commands performed.

scroll default: scroll=½ window

Determines the number of logical lines scrolled when an end-of-file is received from a terminal input in command mode, and the number of lines printed by a command mode *z* command (double the value of *scroll*).

sections default: sections=SHNHH HU

Specifies the section macros for the **[[** and **]]** operations in *open* and *visual.* The pairs of characters in the options's value are the names of the macros which start paragraphs.

shell, **sh** default: sh=/bin/sh

Gives the path name of the shell forked for the shell escape command '!', and by the *shell* command. The default is taken from SHELL in the environment, if present.

shiftwidth, **sw** default: sw=8

Gives the width a software tab stop, used in reverse tabbing with **ˆD** when using *autoindent* to append text, and by the shift commands.

showmatch, sm default: nosm

In *open* and *visual* mode, when a **)** or **}** is typed, move the cursor to the matching **(** or **{** for one second if this matching character is on the screen. Extremely useful with *lisp.*

slowopen, slow terminal dependent

Affects the display algorithm used in *visual* mode, holding off display updating during input of new text to improve throughput when the terminal in use is both slow and unintelligent. See *An Introduction to Display Editing with Vi* for more details.

† 2 for *edit*.

tabstop, ts default: ts=8

The editor expands tabs in the input file to be on *tabstop* boundaries for the purposes of display.

taglength, tl default: tl=0

Tags are not significant beyond this many characters. A value of zero (the default) means that all characters are significant.

tags default: tags=tags /usr/lib/tags

A path of files to be used as tag files for the *tag* command. A requested tag is searched for in the specified files, sequentially. By default, files called **tags** are searched for in the current directory and in /usr/lib (a master file for the entire system).

term from environment TERM

The terminal type of the output device.

terse default: noterse

Shorter error diagnostics are produced for the experienced user.

warn default: warn

Warn if there has been '[No write since last change]' before a '!' command escape.

window default: window=speed dependent

The number of lines in a text window in the *visual* command. The default is 8 at slow speeds (600 baud or less), 16 at medium speed (1200 baud), and the full screen (minus one line) at higher speeds.

w300, w1200, w9600

These are not true options but set **window** only if the speed is slow (300), medium (1200), or high (9600), respectively. They are suitable for an EXINIT and make it easy to change the 8/16/full screen rule.

wrapscan, **ws** default: ws

Searches using the regular expressions in addressing will wrap around past the end of the file.

wrapmargin, **wm** default: wm=0

Defines a margin for automatic wrapover of text during input in *open* and *visual* modes. See *An Introduction to Text Editing with Vi* for details.

writeany, **wa** default: nowa

Inhibit the checks normally made before *write* commands, allowing a write to any file which the system protection mechanism will allow.

10. Acknowledgements

Chuck Haley contributed greatly to the early development of *ex.* Bruce Englar encouraged the redesign which led to *ex* version 1. Bill Joy wrote versions 1 and 2.0 through 2.7, and created the framework that users see in the present editor. Mark Horton added macros and other features and made the editor work on a large number of terminals and Unix systems.