NAME

qed – multi-file text editor

SYNOPSIS

qed [**–**] [**–i**] [**–q**] [**–e**] [**–x** startupfile] [filename ...]

DESCRIPTION

Qed is a multiple-file programmable text editor based on *ed*.

Qed operates on a copy of any file it is editing; changes made in the copy have no effect on the file until a *w* or *W* (write) command is given. The copy of the text being edited resides in a scratch area called a *buffer*. There are 56 buffers, labeled by alphabetics 'a' to 'z' and 'A' to 'Z', and the characters '{', '|', '}' and '~' (the four ASCII characters following 'z'). These 56 characters are called, for notational efficiency, *bnames*. The buffers can contain any character except NUL.

A character is any valid Unicode Codepoint. *Qed* reads and writes UTF–8 encoded text. Arbitrary Unicode Codepoints can also be entered by their hexadecimal value using *Qed*'s special characters '\x', '\u', and '\U' (see below in the **Special Characters** subsection).

If *file* arguments are given, *qed* simulates an *r* command (see below) on each of the named files; that is to say, the files are read into *qed's* buffers so that they can be edited. The first is read into buffer 'a', the second into buffer 'b', through 'z', then from 'A' to 'Z', up to a maximum of 52 files. The optional **–** puts *qed* in non-*verbose* mode (described with the *o* command). The **–q**, **–e** and **–i** are equivalent to performing an initial 'oqs', 'oes' or 'ois' command (see the *o* command below).

When *qed* starts up, the file named by the environment variable **QEDFILE** is read into buffer '~' and executed (i.e. read as command input), before reading in files and accepting commands from the terminal. The argument *filename*s are set in the buffers before the startup file is executed, so the startup file can treat the *filenames* as arguments. The default startup file may be overridden with the **–x** option.

Input to *qed* can be redirected, at any time, to come from storage such as a buffer by use of a *special character* such as "\b". All the *qed special character* sequences are discussed in detail below; they all begin with a backslash '\'.

Qed has a *truth flag* which is set according to the success of certain commands and which can be tested for conditional execution, and a *count* which is set to such values as the number of successful substitutions performed in an *s* command. Each buffer has associated with it a (possibly null) filename and a *changed* flag, which is set if the contents of the buffer are known to differ from the contents of the named file in that buffer.

Commands to *qed* have a simple and regular structure: zero or more *addresses* followed by a single character *command*, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Every command which requires addresses has default addresses, so that the addresses can often be omitted.

In general, any number of commands can appear on a line. Some commands require that the character following the command be a separator, such as blank, tab or newline. Usually, a *display character*, *p*, *P*, *l*, or *L* may precede the separator, causing the resulting line to be displayed in the specified format after the command. Certain commands allow the input of text for placement in the buffer. This text can be supplied in two forms: either on the same line, after the command, or on lines following the command, terminated by a line containing only a period '**.**'. If the text is on the command line, it is separated from the command by a space or a tab. If the tab is used, it is considered part of the text.

Qed supports a limited form of *regular expression* notation. 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. Regular expressions in *qed* are delimited by enclosing them in a pair of identical characters, frequently slashes '/'. In the following specification for regular expressions the word 'character' means any character but newline. Note that special character interpretation always occurs *before* executing a command. Thus, the backslashes mentioned below are those present after special characters have been interpreted.

1. Any character except a metacharacter matches itself. Metacharacters are the regular expression delimiter plus < [**.** and \ | > ^ * + $ when another rule gives them a meaning.

2. A **.** matches any character.

3. A backslash \ followed by any metacharacter in the list given in rule 1 is a regular expression and matches that character. A backslash followed by one of ! _ { } ( ) or a non-zero digit has a special meaning discussed below; otherwise, backslashes have literal meaning in regular expressions.

4. The metacharacter \! matches any control character except tab or newline.

5. A non-empty string *s* enclosed in square brackets [*s*] (or [^*s*]) matches any character in (or not in) *s*. In *s*, \ has no special meaning, and] may only appear as the first character. A substring *a*−*b*, with *a* and *b* in ascending Unicode order, stands for the inclusive range of Unicode characters.

6. A regular expression, of the form <*x1*> or <*x1*|*x2*|...|*xn*>, where the *x*'s are regular expressions of form 1-12, matches what the leftmost successful *x* matches.

7. A backslash followed by a non-zero digit *n* matches a copy of the string that the bracketed regular expression (see rule 11) beginning with the *n*th \(matched.

8. A regular expression of form 1-7 followed by * (+) matches a sequence of zero (one) or more matches of the regular expression.

9. The metacharacter _ matches a non-empty maximal-length sequence of blanks and tabs.

10. The metacharacter \{ (\}) matches the empty string at the beginning (end) of an identifier. An identifier is defined to be an underscore _ or alphabetic followed by zero or more underscores, alphabetics or digits.

11. A regular expression, *x*, of form 1-12, bracketed \(*x*\) matches what *x* matches. The nesting of these brackets in each regular expression of an alternation (rule 6) must be identical. An alternation with these brackets may not be iterated (rule 8).

12. A regular expression of form 1-12, *x*, followed by a regular expression of form 1-11, *y*, matches a match for *x* followed by a match for *y*, with the *x* match being as long as possible while still permitting a *y* match.

13. A regular expression of form 1-12 preceded by ^ (followed by $) is constrained to matches that begin at the left (end at the right) end of a line.

14. A regular expression of form 1-13 picks out the longest among the leftmost matches in a line.

15. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses and the *g* and *v* commands to specify lines, in the *s* command to specify a portion of a line which is to be replaced, in the *G* and *V* commands to refer to buffers in which to perform commands, and in general whenever text is being specified.

To understand addressing in *qed* it is necessary to know that at any time there is a *current buffer* and a *current line.* When *qed* is invoked, the current buffer is buffer 'a', but may be changed at any time by a *b* (change buffer) command. All addresses refer to lines in the current buffer, except for a special case described under the *m* (move) command.

Generally speaking, the current line is the last line affected by a command; however, the exact effect on the current line is discussed under the description of the command. Addresses are constructed as follows.

1. The character '**.**' addresses the current line.

2. The character '$' addresses the last line of the buffer.

3. A decimal number *n* addresses the *n*-th line of the buffer.

4. ''*x*' addresses the line marked with the mark name character *x*, which must be a bname. Lines are marked with the *k* command described below. It is an error for the marked line to be outside of the current buffer.

5. A regular expression enclosed in slashes '/' addresses the first matching line found by searching forwards from the line after the current line. If necessary, the search wraps around to the beginning of the buffer. If the trailing '/' would be followed by a newline, it may be omitted.

6. A regular expression enclosed in queries '?' addresses the first matching line found by searching backwards from the line before the current line. If necessary the search wraps around to the end of the buffer. If the trailing '?' would be followed by a newline, it may be omitted.

7. An address followed by a plus sign '+' or a minus sign '−' followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.

8. An address followed by '+' or '−' followed by a regular expression enclosed in slashes specifies the first matching line following (resp. preceding) that address. The search wraps around if necessary. The '+' may be omitted. Enclosing the regular expression in '?' reverses the search direction.

9. If an address begins with '+' or '−' the addition or subtraction is taken with respect to the current line; e.g. '−5' is understood to mean '**.**−5'.

10. If an address ends with a '+' (or '−') 1 is added (resp. subtracted). As a consequence of this rule and rule 9, the address '−' refers to the line before the current line. Moreover, trailing '+' and '−' characters have cumulative effect, so '−−' refers to the current line less 2.

11. To maintain compatibility with earlier versions of the editor, the character '^' in addresses is entirely equivalent to '−'.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when none is given. If more addresses are given than the command requires, the last one or two (depending on what is accepted) are used. The last addressed line must not precede the second-last addressed line.

Typically, addresses are separated from each other by a comma ','. They may instead be separated by a semicolon ';' in which case the current line '**.**' is set to the first address before the second address is interpreted. The second of two separated addresses may not be a line earlier in the buffer than the first. If the address on the left (right) side of a comma or semicolon is absent, it defaults to the first (resp. last) line.

Filename operands of commands may be made up of printing characters only. However, when the filename appears as the argument to the invocation of *qed*, non-printing characters may be included. When a filename is specified for a command, it is terminated at the first blank, tab or newline.

In the following list of *qed* commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the default.

(**.**) a <text>

The append command accepts input text and appends it after the addressed line. '**.**' is left on the last line input, if there were any, otherwise at the addressed line. Address '0' is legal for this command; text is placed at the beginning of the buffer.

b<bname>

The change buffer command sets the current buffer to be that named. '**.**', '$' and the remembered *filename* are set to those of the new buffer; upon return to a previously used buffer, '**.**' will be set to its value when the buffer was last used.

(**.**) b[+−**.**][pagesize][display character]

The browse command provides page-oriented printing. The optional '+', '−', or '**.**' specifies whether the next, previous, or surrounding page is to be printed; if absent, '+' is assumed. *b***.** also prints several carets '^^^^^' immediately below the current line. If a pagesize is given, it is used for the current browse command and remembered as the default. The pagesize is initially 22 lines. If a display character is given, the lines are printed in the specified format, and the format is remembered as the default. Initially, 'p' is the default. For *b*+ and *b*−, '**.**' is left at the last line displayed; for *b***.**, it is unchanged. NOTE: The browse and change buffer commands are the same character! The two commands can be syntactically distinguished in all cases except for 'b<display char>'; this ambiguity may be resolved by typing the (implicit) '+' after the 'b'.

(. , .) c <text>

The change command deletes the addressed lines, then accepts input text which replaces these lines. '.' is left at the last line input; if there were none, it is left at the line preceding the deleted lines. If an interrupt signal (usually ASCII DEL) is received during a change command, the old lines are not deleted.

(. , .) d

The delete command deletes the addressed lines from the buffer. The line after the deleted section becomes the current line; if the deleted lines were originally at the end, the new last line becomes the current line. The character after the 'd' can only be one of a blank, newline, tab, or display character. Line 0 is a valid address for deletion; deleting line 0 has no effect on any lines in the buffer.

e filename

The edit command causes the entire contents of the current buffer to be deleted, and then the named file to be read in. '.' is set to the last line of the buffer. The number of characters read is typed if *qed* is in *verbose* mode. The *filename* is remembered for possible use as a default file name in a subsequent *f*, *r*, *w*, or *W* command.

E filename

The *E* command is like *e*, except that *qed* does not check to see if the buffer has been modified since the last *w* command.

f filename

The filename command prints information about the current buffer, in the format used by the *n* command. If *filename* is given, the currently remembered file name is changed to *filename*. If *qed* is not in verbose mode, the information is only printed if the *filename* is not specified. If it is not desired to set the *filename,* the character immediately after the f must be a newline. Otherwise, the first token (which may be the null string) on the line, after a mandatory non-empty sequence of blanks and tabs, is taken to be the *filename.* These rules apply to all *filename*-using commands, *e, f, r, R, S, w* and *W,* although some regard specification of an explicitly null *filename* as an error.

(1 , $) g/regular expression/command list

In the global command, the first step is to mark every line in the range which matches the regular expression. Then for every such line, the command list is executed with '.' initially set to that line. Any embedded newlines in the command list must be escaped with a backslash. The *a*, *i*, and *c* commands and associated input are permitted; the '.' terminating input mode may be omitted if it would be on the last line of the command list. The commands *g* and *v* are not permitted in the command list. If the command list is empty, '.p' is assumed. The regular expression may be delimited by any character other than newline.

G/regular expression/command list

In the globuf command, the first step is to mark every active buffer whose output from an *f* command (with the *filename* printed literally) would match the regular expression. (An active buffer is one which has either some text or a remembered file name.) Then for every such buffer, the command list is executed with the current buffer set to that buffer. In other respects it is like the global command, except that only the commands *G* and *V* are not permitted in the command list. If the command list is empty, 'f' is assumed.

h<option> command list

The until command provides a simple looping mechanism. The command list is a newline-terminated command sequence which forms the body of the loop; embedded newlines must be escaped with a backslash. The option specifies the exit condition for the loop, and is specified by the character(s) immediately following the 'h':

h[*N*]t The loop is executed until the truth flag is true.
h[*N*]f The loop is executed until the truth flag is false.
h[*N*] The loop is executed indefinitely.

The loop condition is tested *after* execution, so the 'ht' and 'hf' forms execute at least once. *N* denotes an optional non-negative number which indicates the maximum number of times to execute

the loop.

(**.**) i <text>
The insert command accepts input text and inserts it before the addressed line. '**.**' is left at the last line input; if there were none, at the line before the addressed line. This command differs from the *a* command only in the placement of the text.

(**.**–1 , **.**) j
(**.**–1 , **.**) j /replacement/
The join command collapses all addressed lines into a single line by deleting intermediate newlines. The *replacement* (if any) is placed between joined lines. Newlines, backslashes '\', and slashes '/' within *replacement* must be preceded by a backslash. Only slashes may delimit *replacement*. '**.**' is left at the resulting line. NOTE: The join command in *qed* has a different default addressing from that in *ed.*

(**.**) k<bname>
The mark command marks the addressed line with the given bname. (The bname used in the mark has no relation to any buffer; it is just a label.) The address form '´<bname>' then addresses this line. '**.**' is not changed. The marks are global to *qed*; marking a line 'x' erases any previous mark 'x' in any buffer.

(**.** , **.**) l
The list command prints the addressed lines in an unambiguous way: a tab is printed as '\t', a backspace as '\b', a backslash as '\\'. Non-printing characters in the ASCII range are printed as '\x*hh*'; all characters in the Basic Multilingual Plane are printed as '\u*hhhh*', and characters beyond the Basic Multilingual Plane are printed as '\U*hhhhhh*', for *h...* (lowercase) hexadecimal digits. A long line is folded, with the second and subsequent sub-lines indented one tab stop. If the last character in the line is a blank, it is followed by '\n'.

(**.** , **.**) L
The *L* command is similar to the *l* command, but each line displayed is preceded by its line number, any marks it has (which appear as '´*x*'), and a tab.

(**.** , **.**) m*a*
The move command repositions the addressed lines after the line addressed by *a*. The last of the moved lines becomes the current line. The address *a* can also be of the form <bname>address, in which case the text is moved after the address in the named buffer. The buffer to which the text was moved becomes the current buffer. The original buffer (if different) has '**.**' left at the line before the moved lines.

n The names command displays the bname, dollar and *filename* (in 'l' format) of the current buffer and all active buffers. If the buffer's changed flag is set, an apostrophe '´' is printed after the bname. The current buffer is indicated by a period '**.**' before the dollar value. If present, the *filename* is preceded by a tab.

N The *N* command is similar to the *n* command, but the display is only given for those buffers which have a *filename* and for which the changed flag is set.

o*ps* The option command allows various options to be set. The first argument, *p*, specifies which option is being set. The rest of the command, *s*, specifies the setting. Most options can be either enabled or disabled; *s* is 's' to set the option, or 'r' to reset it. The following table describes the available options. The default setting is shown after the option's letter.

b22p Set the length and format of the page printed by the browse command. Either the length or the format may be omitted.

B<null string>
Set the default command sequence to be performed when a newline command is typed at the terminal. The command sequence is set by following the 'B' with a newline-terminated string. If the string is null, the newline command resumes its default behaviour.

cr Set the changed flag of the current buffer.

dr Dualcase search mode affects rule one of regular expression construction so that a letter is matched without regard to its case.

er Error exit mode causes *qed* to exit if an error occurs (see the DIAGNOSTICS section). This option is mainly intended for use of qed in shell files.

ir Interrupt catching mode causes *qed* to exit when interrupted. (This includes removing the temporary file).

pr Prompting mode causes '*' to be typed immediately before a command (as opposed to text) is read from the terminal.

qr Quit catching mode causes *qed* to dump core, leaving the temporary file intact, when a QUIT signal is received.

Tr Tracing mode causes all commands not typed directly by the user to be echoed on the terminal. When a special character (other than '\B or '\N') is encountered, a '[' is typed, followed by a code specifying the character — 'za' for register 'a', 'g' for global command list, 'l' for '\l', 'B' for browse pseudo-register, etc. Then, an '=' is typed, followed by the interpretation of the special character, followed by a ']'.

us Uppercase conversion mode enables case transformation in substitute commands. If the 'u' flag is set, the character caret ('^') becomes non-literal in the replacement text of a substitution. It behaves just like '&', but with case switching of alphabetics in the replaced text. If the flag is 'u', all alphabetics are mapped to upper case; if 'l', lower case; and if 's', the case is switched.

vs Verbose mode causes character counts to be typed after *e*, *r*, *w*, *R*, *S*, and *W* commands. It also causes '!' to be typed upon completion of the *!*, <, | and > commands.

?*c* *c* must be one of 'c', 'd', 'i', 'p', 'T' or 'v'. The value of the corresponding flag is stored in the truth.

(**.** , **.**) p

The print command prints the addressed lines. '**.**' is left at the last line printed.

(**.** , **.**) P

The PRINT command is similar to the print command, but each line displayed is preceded by its line number, any marks it has (which appear as '´*x*'), and a tab.

q The quit command causes *qed* to exit. No automatic write of a file is done. If the changed flag is set in any buffer, *qed* prints '?q' and refuses to quit. A second *q* or a *Q* will get out regardless, as will an end-of-file on the standard input.

Q Like *q*, but changed flags are not checked.

($) r filename

The read command reads in the given file after the addressed line. If no *filename* is given, the remembered *filename* is used (see *e* and *f* commands). The *filename* is remembered if there was not already a remembered *filename* in the current buffer. Address '0' is legal for *r* and causes the file to be read at the beginning of the buffer. If *qed* is in *verbose* mode and the read is successful, the number of characters read is typed, except while *qed* is starting up, in which case an *f* command is performed. '**.**' is left at the last line read in from the file.

R filename

The restore command restores an environment saved by a save (*S*) command. The changed flag in each buffer is restored from the files; all other flags are unaffected. The input stack is reset to the top (teletype input) level, and the current buffer becomes 'a'. '**.**' is left at the saved value of '**.**' in buffer 'a'**.** If the *filename* is not specified, 'q' is used.

(**.** , **.**) s*n*/regular expression/replacement/

(**.** , **.**) s*n*/regular expression/replacement/g

The substitute command searches each addressed line for occurrences of the specified regular expression. The decimal number *n* defaults to 1 if missing. On each line in which *n* matches are found, the *n*th matched string is replaced with *replacement*. If the global replacement indicator 'g' follows the command, all subsequent matches on the line are also replaced. Within a line, a search starts from

the character following the last match, unless the last match was an empty string, in which case the search starts at the second character following the empty string (to ensure a match is not repeated). It is an error for the substitution to fail on all addressed lines unless it is in a global command. '**.**' is left at the last line substituted.

Any character other than newline or a numeral may be used instead of '/' to delimit the regular expression and *replacement*. If the trailing delimiter is missing (i.e., an unescaped newline in the *replacement*), its presence is assumed, and the last line affected is printed, as if the substitute was followed by a *p* command. If delimiter following the expression is omitted as well, an empty *replacement* is assumed.

An ampersand '&' appearing in *replacement* is replaced by the string matching the regular expression. As a more general feature, the characters '*n*', where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression enclosed between '\(' and '\)'. When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left.

A caret '^' appearing in *replacement* behaves much like an ampersand, but provides a mechanism for case switching of alphabetics, as discussed under the *o* command. To include an ampersand '&', caret '^', backslash '\', newline, or the delimiter literally in *replacement*, the character must be preceded by a backslash. Lines may be split by substituting newline characters into them.

S filename

The save command saves the full buffer and register information in two files called 'filename:aq' and 'filename:bq'. If the filename is absent, 'q' is used. If the filename has more than 12 characters after the last slash '/', it is truncated to 12 characters to avoid overwriting the file.

(**.** , **.**) t*a*

The copy command acts just like the move *m* command except that a copy of the addressed lines is placed after address *a*. '**.**' is left on the last line of the copy. The buffer to which the text was copied becomes the current buffer.

u The undo command restores the last line changed by a *s*, *u*, or *x* command. Any new lines created by splitting the original are left. It is an error if the line is not in the current buffer. '**.**' is left at the restored line.

(1 , $) v/regular expression/command list

This command is the same as the global command except that the command list is executed with '**.**' initially set to every line *except* those matching the regular expression.

V/regular expression/command list

This command is the same as the globuf command except that the command list is executed with the current buffer initially set to every active buffer *except* those matching the regular expression.

(1 , $) w filename

The write command writes the addressed lines onto the given file. If the file does not exist, it is created. The filename is remembered if there was not already a remembered file name in the current buffer. If no file name is given, the remembered file name is used. '**.**' is unchanged. If *qed* is in *verbose* mode and the command is successful, the number of characters written is typed.

(1 , $) W

The *W* command is the same as the *w* command except that the addressed lines are appended to the file.

(**.** , **.**) x

The xform command allows one line at a time to be modified according to graphical requests. The line to be modified is typed out, and then the modify request is read from the terminal (even if the xform command is in a global command or other nested input source). Generally each character in the request specifies how to modify the character immediately above it, in the original line, as described in the following table.

Delete the above character.

% Replace the above character with a space.

^ Insert the rest of the request line before the above character. If the rest of the request line is empty, insert a newline character.

$ Delete the characters in the above line from this position on; replace them with the rest of the request line.

space or tab:
Leave above character(s) unchanged.

any other:
This character replaces the one above it.

If the request line is longer than the line to be modified, the overhang is added to the end of the line without interpretation, that is, without treating '#', '%', '^' or '$' specially. Any characters after a '^' or '$' request are not interpreted either.

Xform will not process control characters other than tab and newline, except in contexts where it need not know their width (that is, after a '^' or '$' request, or in the part of either the request or the line that overhangs the other). Remember that the ERASE character (processed by the system) erases the last character typed, not the last column.

Some characters take more than one column of the terminal to enter or display. For example, entering the ERASE or KILL characters literally takes two columns because they must be escaped. To delete a multi-column character, one must type '#' under all its columns. To replace a multicolumn character, the replacement must be typed under the first column of the character. Similarly, if a replacement character is multi-columned, it replaces the character in its first column.

The tab character prints as a sequence of spaces, and may be modified as if it were that sequence. As long as the last space is unmodified, it and the remaining contiguous spaces will represent a tab.

The modification process is repeated until the request is empty. Only a newline may immediately follow the 'x'.

y<condition><type>

The jump command controls execution nested input sources. The condition is compared to the truth flag to see if the jump should be performed; if a 't', the jump is performed if the truth flag is true, if an 'f', the jump is performed if the truth flag is false, if absent the jump is always performed. Several types of jumps exist:

y[tf]o Jump out of the current input source. If the current input source is the command line for a *g*, *G*, *v*, *V* or *h* command, the command is terminated.

y[tf]*N* Control is transferred to absolute line *N* (an integer) in the executing buffer. The current input source must be a buffer.

y[tf]´<label>
Control is transferred to the first line found, searching forward in the buffer, that begins with a comment "<label>. The match of the labels must be exact; regular expressions are not used to define the control label. (A tab, blank or newline after the double quote specifies a null label: a line beginning '" LAB' cannot be transferred to by this form of jump.) If no such label is found, control resumes at the character after the label in the jump command. The current input source must be a buffer.

y[tf]`<label>
Similar to 'y´<label>', but the search is in the opposite (reverse) direction.

y[tf] If no recognized type is given, input is skipped up to the next newline.

It is an error if reading the label or line number for a jump command causes the current input source (i.e. buffer) to be 'popped.' This can happen if the label is the last word in the buffer, but can be circumvented by putting an extra blank or newline after the jump command.

(**.** , **.**) z*Xc*

Qed has 56 registers labeled by bnames. Three of these, registers 'T', 'C', and 'U', are reserved: 'T' is the truth flag, 'C' is the count, 'U' contains the UNIX command from the most recent bang, crunch,

zap, or pipe command. The contents of register *X*, where *X* is a bname, can be inserted into the input stream with the special character ''\z*X*''. The command ''z*X*'' specifies register *X* as the argument to the operation character (signified above by *c*) that follows it. In the description below, *N* stands for a possibly signed decimal integer and *S* stands for a newline-terminated string. Newlines may be embedded in registers by escaping them with a backslash. Although some of the register commands refer to addressed lines, '**.**' is unaffected by a *z* command. The operations are as follows:

p Print the contents of the register in 'p' format.

l Print the contents of the register in 'l' format.

. Set the register to the contents of the addressed line.

/reg-exp/
Set the register to the portion of current line that matches the regular expression in slashes. If no such pattern is found, the register is cleared. The truth flag is set according to whether a match was found.

:*S* Set register to the string following the colon.

´*Y* Make a direct copy of register *Y* in register *X*, without interpreting special characters. *Y* is any register bname.

+*N* Increment by *N* the Unicode value of each character in the register. Similarly, a '−' decrements each character.

=*S* (Or '<' or '>' or '!=' or '!<' or '!>'.) Set truth flag to the result of the lexical comparison of the register and the string *S*.

n Set the count to the length of the register.

)*N* (Or '('.) 'Take' the first *N* characters of the register, i.e. truncate at the *N*+1'th character. '(' ('drop') is the complementary operator; it deletes the first *N* characters from the register. If *N* is negative, the break point is $|N|$ from the end.

[/reg-exp/
Set the count to the starting index of the regular expression in the register. Set the truth to whether the expression matches any of the register.

s*n*/reg-exp/replacement/
s*n*/reg-exp/replacement/g
Perform a substitute command with semantics identical to the *s* command, but in the text of the register, not a line of the buffer.

C 'Clean' the register: collapse each occurrence of 'white space' in the register to a single blank, and delete initial and trailing white space.

{ *S* Set the register to the value of the shell environment variable *S*, whose name may be terminated by a space, tab, newline or '}'.

The registers can also be manipulated as decimal numbers. Numerical operations are indicated by a number sign '#' after the register name: e.g. 'zx#+2'. It is an error to attempt to perform arithmetic on a register containing non-numeric text other than a leading minus sign. The numerical operations are:

a Set the value of the register to be the value of the address given to the command; e.g. '$za#a' sets register 'a' to the number of lines in the buffer.

r Set register *X* to be the first address given the command, and *X*+1 to be the second. If *X* is '~', an error is generated. For example, '5,$zi#r' sets register 'i' to 5, and register 'j' to the value of '$'. '**.**' is unchanged. This command is usually used to pass addresses to a command buffer.

n Set register to the length of the addressed line.

:*N* Set register to *N*. Scanning of the number stops at the first non-numeric character, not at the end of the line.

+*N* Increment register by *N*. '−', '*', '/', and '%' decrement, multiply, divide, or modulo the register by *N*.

p Print current contents of register.

P Set register to the decimal value of the process id of *qed*.

=*N* (Or '<' or '>' or '!=' or '!<' or '!>'.) Set truth flag to the result of the numeric comparison of the register and the number *N*.

Several numerical operations may be combined in one command (and it is more efficient to do so when possible.) For example, '$zd#a−3' sets register 'd' to three less than the value of '$'.

Z The zero command clears the current buffer. The contents, filename and all flags for the buffer are zeroed. The character after the 'Z' must be a blank, tab or newline.

($) =
The line number of the addressed line is typed. '**.**' is unchanged.

! <UNIX command>
The bang command sends the command line after the '!' to the UNIX shell to be interpreted as a command. Embedded newlines must be preceded by a backslash. The signals INTR, QUIT, and HUP are enabled or disabled as on entry to *qed*. At the completion of the command, if *qed* is in *verbose* mode, an '!' is typed. The return status of the command is stored in the truth flag. '**.**' is unchanged.

The command line is stored in register 'U'. If a second '!' immediately follows the first, it is replaced with the uninterpreted contents of this register. Thus '!!' repeats the most recent bang command, and '!! | wc' repeats the command with an additional pipeline element added.

(1 , $) > <UNIX command>
The zap command is similar to the bang command, but the addressed lines become the default standard input of the command. The command is stored in register 'U', as for bang; '>>' corresponds to '!!'.

($) < <UNIX command>
The crunch command is similar to the bang command, but the standard output of the command is appended to the current buffer after the addressed line, as though read with an *r* command from a temporary file. The command is stored in register 'U' as for bang; '<<' corresponds to '!!'. '**.**' is left at the last line read.

(1 , $) | <UNIX command>
The pipe command is similar to the bang command, but the addressed lines become the default standard input of the command, and are replaced by the standard output of the command. The command is stored in register 'U' as for bang; '||' corresponds to '!!'. If the command returns non-zero status, the original lines are not deleted. '**.**' is left at the last line read.

(**.**)" The comment command sets dot to the addressed line, and ignores the rest of the line up to the first following double quote or newline. If, however, the character immediately after the double quote is a second double quote (i.e. the command is ""), the text which would normally be ignored is typed on the standard output. Special characters in the text will be interpreted, whether or not the text is printed, so to print a message such as "Type \bx" requires the command "" "Type \cbx". Commented lines are used as labels by the *y* (jump) command.

% The register print command displays the name and value of all defined registers, followed by the \p ('P') and \r ('R') pseudo-registers, and the browse ('B') pseudo-register, if defined.

The numeric register print command displays the name and value of all defined registers with numeric values.

(**.**+1 , **.**+1)<newline>
An address or addresses alone on a line cause the addressed lines to be printed. If the last address separator before the newline was ';', only the final addressed line is printed. A blank line alone causes the contents of the browse pseudo-register (described with the *o* command) to be executed. If the register is null, as it is initially, the newline command behaves as though the register contains '**.**+1p'.

Special Characters

Qed has some special character sequences with non-literal interpretations. These sequences are processed

at the *lowest* level of input, so their interpretation is completely transparent to the actual commands. Whenever input from the user is expected, a special character can appear and will be processed. Special characters can be nested in the sense that, for example, a buffer invoked by '\b' can contain a register invocation '\z'. Backslashed escape sequences such as '\(' in regular expressions are *not* special characters, so are not interpreted at input. The sequence '\(' is left untouched by the input mechanism of *qed;* any special meaning it receives is given it during regular expression processing. The special characters are:

\b<bname>
: The 'b' must be followed by a bname. When '\b*X*' is typed, the contents of buffer *X*, up to but *not including* the last newline, are read as if they were entered from the keyboard. Typically, the missing newline is replaced by the newline which appears after the buffer invocation. Changing the contents of an executing buffer may have bizarre effects.

\B
: Equivalent to current buffer's bname.

\c
: The sequence \c is replaced by a single backslash, which is not re-scanned. The effect of the 'c' is to delay interpretation of a special character.

\f
: Equivalent to current buffer's file name.

\F<bname>
: Equivalent to the file name in the named buffer.

\l
: One line is read from the standard input up to, but *not including* the terminal newline, which is discarded. Note that the first invocation will read the remainder of the last line entered from the keyboard. For example, if a buffer is invoked by typing the line:

```
\bxjunk
```

the first \l in buffer 'x' will return the string 'junk'.

\N
: Equivalent to a newline. Primarily useful when delayed.

\p
: Equivalent to the most recent regular expression used.

\r
: Equivalent to the replacement text of the most recent substitute or join command.

\u<hhhh>
: Inserts the Unicode Codepoint 0xhhhh. <hhhh> must be exactly four (lowercase) hexadecimal digits. If <hhhh> is malformed, or is not a valid Unicode Codepoint, error '?U' is generated.

\U<hhhhhh>
: Inserts the Unicode Codepoint 0xhhhhhh. <hhhhhh> must be exactly six (lowercase) hexadecimal digits. If <hhhhhh> is malformed, or is not a valid Unicode Codepoint, error '?U' is generated.

\x<hh>
: Inserts the ASCII character/Unicode Codepoint 0xhh. <hh> must be exactly two (lowercase) hexadecimal digits. If <hh> is malformed, or is not a valid Unicode Codepoint, error '?U' is generated.

\z<bname>
: Equivalent to the contents of register '\z*X*'. If the register changes during execution, the changes appear immediately and affect execution. If a '+' ('−') appears between the 'z' and the bname, the Unicode values of the characters in the register are incremented (decremented) by one before interpretation. If a '#' precedes the '+' ('−') the contents of the register are numerically incremented (decremented).

\"
: The sequence \" means 'no character at all'! It is primarily used to delay interpretation of a period that terminates an append, until the second or third time it is read (e.g. in loading execution buffers): the sequence \c"**.** at the beginning of a line puts a period on the line which will terminate an append the second time it is read.

\´[bfFlprz]
: If an apostrophe appears between the backslash and the identifying character for one of the special characters '\b', '\f', '\F', '\l', '\p', '\r' or '\z', interpretation is as usual except that any further special characters *embedded* in the buffer, register, etc. are *not* interpreted. Actually, any special character may be quoted, but in forms such as '\´B', the quote has no effect.

A special character is interpreted immediately when it appears in the input stream, whether it is currently coming from the teletype, a buffer, a register, etc. (This includes characters read when typing a special

character: '\´b\za', with register 'a' containing the character 'X', invokes the literal contents of buffer 'X'.) Thus, interpretation is recursive unless the special character is '\c'. Special characters appearing in text processed in a command such as move, read or write, are *not* interpreted. If the backslash-character pair is not a special character from the above list, it is passed unchanged. Interpretation may be delayed using '\c'; for example, if a '\bx' is to be appended to a buffer for later interpretation, the user must type '\cbx'. To delay interpretation *n* times, *n* c's must be placed between the backslash and the identifying character. In regular expressions and substitutes, a backslash preceding a metacharacter turns off its special meaning. Even in these cases, a backslash preceding an ordinary character is not deleted, unlike in *ed*. For example, since the 'g' command must read its entire line, a '\zx' in a substitute driven by a global must be delayed if the contents of the register are to be different for each line, but since '\&' is not a special character except to the substitute, its interpretation need not be delayed:

```
zA#:1
g/\$/ s/\.xyz/\czA \&/p  zA#+1
```

globally searches for lines with a literal currency sign, and on each one substitutes for '.xyz' the contents of register 'A' at the time of substitution, followed by a space and a literal ampersand, prints the result and increments register 'A'. As a second example, the substitute

```
s/xyz/\\N&/
```

replaces 'xyz' with a newline followed by 'xyz'. Note that the '\\N' is interpreted as 'backslash followed by newline,' as the sequence '\\' *has no special meaning in qed* outside of regular expressions and replacement text. However, to match, say, '\\z' using a regular expression, it must be entered as '\\\cz'.

If an interrupt signal (ASCII DEL) is sent, *qed* prints '??' and returns to its command level. If a hangup signal is received, *qed* executes the command 'S qed.hup'.

Some size limitations: 512 characters per line, 256 characters per global command list, 1024 characters of string storage area, used for storing registers, file names and regular expressions, 16 levels of input nesting, and 128K characters in the temporary file. The limit on the number of lines depends on the amount of core: each line takes 1 word.

FILES

Qed creates a temporary file */tmp/qXXXXXX* by calling the C Standard Library function *mkstemp(3)*.

Files of the form <name>*:aq* and <name>*:bq* are created when the 'S <name>' command is issued. If the editor receives a SIGHUP signal, it creates the save files 'qed.hup:aq' and 'qed.hup:bq'. The '*:[ab]q' files can be read back in with 'R <name>'.

DIAGNOSTICS

Diagnostics are in the form of '?' followed by a single letter code. If the diagnostic is because of an inaccessible file, the offending file name is also displayed. If input is not from the highest level (i.e. the standard input, usually the terminal), a traceback is printed, starting with the lowest level. The elements of the traceback are of the form ?b*XM.N* or ?z*XN*, where *X* is the buffer or register being executed when the error was encountered, *M* is the line number in the buffer and *N* is the character number in the line or register. The possible errors are:

0	non-zero status return in | command
F	bad bname for \F
G	nested globuf commands
N	last line of input did not end with newline
O	unknown option in the *o?c* command
R	restore (*R*) command failed (file not found or bad format)
T	I/O error or overflow in tempfile
U	malformed or invalid Unicode codepoint
Z	out of string space; clear a few registers or file names

a	address syntax
b	bad bname in a *b* command or for \b
c	ran out of core
f	filename syntax error
g	nested global commands
i	more than 52 files in initialization argument list
k	bad bname in *k* command
l	an internal table length was exceeded
m	tried to move to an illegal place (e.g. 1,6m4)
o	error opening or creating a file
p	bad regular expression (pattern) syntax
q	*e* with the current changed flag set, or *q* with any changed flag set
r	read error from file
s	no substitutions found
t	bad *x* command data or single-case terminal
u	no line for *u* command to undo
x	command syntax error
w	write error on file
y	bad jump command (including popping the input buffer while scanning the label)
z	bad register bname
|	failure to create pipe for <, | or > command
#	bad numeric register operation
$	line address out of range
?	interrupt
/	line search failed
[	bad index in a register take or drop command
\	attempt to recursively append a buffer
!	jackpot — you found a bug in regular expression matching

SEE ALSO

The source code for this version of *Qed* at https://github.com/phonologus/QED

Arnold Robbins' *Qed Archive* at https://github.com/arnoldrobbins/qed-archive

A Tutorial Introduction to the ED Text Editor (B. W. Kernighan)

Programming in *Qed*: a Tutorial (Robert Pike)

ed(1)

HISTORY

Written at U of T, based on several incarnations of *ed,* with contributions from Tom Duff, Robert Pike, Hugh Redelmeier and David Tilbrook.

Updated to work with UTF−8 and Unicode in 2020 by Sean Jensen.

BUGS

The changed flag is not omniscient; changing the contents of the file outside of *Qed* will fool it.

Xform *could* work on single-case terminals, but backslashes become very confusing for the user.

Xform will struggle with right-to-left, bi-directional, and wide-character lines. It's use should be confined to single-width, left-to-right lines, otherwise chaos will likely ensue.

If *Qed* reads in a file that does not terminate in a newline, it will add one, and raise a '?N: newline added' warning. This behaviour differs from previous versions of *Qed*, but agrees with the behaviour of **Plan9** *ed*. This may actually be a feature.

On the PDP-11, numeric registers are 16-bit integers, but the count is a 32-bit integer.