ed

COLLABORATORS			
	TITLE : ed		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 7, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ed

1.1 ed.guide

This info file documents the ‘ed’ text editor, as of release 0.2.
You may find in this document:

1.2 ed.guide/Intro

Intro
*****

‘ed’ is a line-oriented text editor. It is used to create, display, modify and otherwise manipulate text files, both interactively and via shell scripts. A restricted version of ‘ed’, ‘red’, can only edit files in the current directory and cannot execute shell commands. ‘ed’ is the "standard" text editor in the sense that it is the original editor for Unix, and thus widely available. For most purposes, however, it is superceded by full-screen editors such as Emacs and Vi.

The sample sessions below illustrate some basic concepts of line editing with ‘ed’. We begin by creating a file, ‘sonnet’, with some help from Shakespeare. As with the shell, all input to ‘ed’ must be followed by a newline character. Comments begin with a ‘#’.

```
$ ed
# The ‘a’ command is for appending text to the editor buffer.
a
No more be grieved at that which thou hast done.
```

```
Roses have thorns, and filvers foutians mud.
Clouds and eclipses stain both moon and sun,
And loathsome canker lives in sweetest bud.
.
# Entering a single period on a line returns 'ed' to command mode.
# Now write the buffer to the file 'sonnet' and quit:
w sonnet
183
# 'ed' reports the number of characters written.
q
$ ls -l
total 2
-rw-rw-r--    1 alm           183 Nov 10 01:16 sonnet
$
```

Editing with 'ed' is done in two distinct modes: "command" and "input". When first invoked, 'ed' is in command mode. In this mode commands are read from the standard input and executed to manipulate the contents of the editor buffer. When an input command, such as 'a' (append), 'i' (insert) or 'c' (change), is given, 'ed' enters input mode. This is the primary means of adding text to a file. In this mode, no commands are available; instead, the standard input is written directly to the editor buffer. A "line" consists of the text up to and including a newline character. Input mode is terminated by entering a single period ('.') on a line.

In the next example, some typos are corrected in the file 'sonnet'.

```
$ ed sonnet
183
# Begin by printing the buffer to the terminal with the 'p' command.
# The ',' means ''all lines.''
,p
No more be grieved at that which thou hast done.
Roses have thorns, and filvers foutians mud.
Clouds and eclipses stain both moon and sun,
And loathsome canker lives in sweetest bud.
# Select line 2 for editing.
2
Roses have thorns, and filvers foutians mud.
# Use the substitute command, 's', to replace 'filvers' with 'silver',
# and print the result.
s/filvers/silver/p
Roses have thorns, and silver foutians mud.
# And correct the spelling of 'fountains'.
s/utia/untai/p
Roses have thorns, and silver fountains mud.
w sonnet
183
q
$
```

'ed' may be invoked with or without arguments See Invoking ed. When invoked with a FILE argument, a copy of FILE is read into the editor's buffer. Changes are made to this copy and not directly to FILE itself. Upon quitting 'ed', any changes not explicitly saved with a 'w' command See Commands, are lost.

Since ‘ed’ is line-oriented, we have to tell it which line, or range of lines we want to edit. In the above example, we do this by specifying the line’s number, or sequence in the buffer. Alternatively, we could have specified a unique string in the line, e.g., ‘/filvers/’, where the ‘/’s delimit the string in question. Subsequent commands affect only the selected line, a.k.a. the "current" line. Portions of that line are then replaced with the substitute command, whose syntax is ‘s/OLD/NEW/’.

Although ‘ed’ accepts only one command per line, the print command ‘p’ is an exception, and may be appended to the end of most commands.

In the next example, a title is added to our sonnet.

```
$ ed sonnet
183
a
 Sonnet #50
.
,p
No more be grieved at that which thou hast done.
Roses have thorns, and silver fountains mud.
Clouds and eclipses stain both moon and sun,
And loathsome canker lives in sweetest bud.
 Sonnet #50
# The title got appended to the end; we should have used ‘0a’
# to append ‘‘before the first line.’’
# Move the title to its proper place.
5m0p
 Sonnet #50
# The title is now the first line, and the current line has been
# set to this line as well.
,p
 Sonnet #50
No more be grieved at that which thou hast done.
Roses have thorns, and silver fountains mud.
Clouds and eclipses stain both moon and sun,
And loathsome canker lives in sweetest bud.
wq sonnet
195
$
```

When ‘ed’ opens a file, the current line is initially set to the last line of that file. Similarly, the move command ‘m’ sets the current line to the last line moved.

In summary: All ‘ed’ commands operate on whole lines or ranges of lines; e.g., the ‘d’ command deletes lines; the ‘m’ command moves lines, and so on. It is possible to modify only a portion of a line by means of replacement, as in the second example above. However even there, the ‘s’ command is applied to whole lines at a time.

Structurally, ‘ed’ commands consist of zero or more line addresses, followed by a single character command and possibly additional parameters; i.e., commands have the structure:

```
     [ADDRESS [,ADDRESS]]COMMAND[PARAMETERS]
```

   The ADDRESS(es) indicate the line or range of lines to be affected by the command. If fewer addresses are given than the command accepts, then default addresses are supplied.

   Related programs or routines are ‘vi (1)’, ‘sed (1)’, ‘regex (3)’, ‘sh (1)’. Relevant documents are:

     Unix User’s Manual Supplementary Documents: 12 -- 13

     B. W. Kernighan and P. J. Plauger: "Software Tools in Pascal", Addison-Wesley, 1981.

1.3 ed.guide/Invoking ed

```
Invoking GNU ‘ed’
*****************

     ed [-] [-Gs] [-p STRING] [FILE]
     red [-] [-Gs] [-p STRING] [FILE]

‘-G’
     Forces backwards compatibility. This affects the behavior of the
     ‘ed’ commands ‘G’, ‘V’, ‘f’, ‘l’, ‘m’, ‘t’ and ‘!!’. If the
     default behavior of these commands does not seem familiar, then
     try invoking ‘ed’ with this switch.

‘-s’
‘-’
     Suppresses diagnostics. This should be used if ‘ed’’s standard
     input is from a script.

‘-p STRING’
     Specifies a command prompt. This may be toggled on and off with
     the ‘P’ command.

   FILE specifies the name of a file to read. If FILE is prefixed with
a bang (!), then it is interpreted as a shell command. In this case,
what is read is the standard output of FILE executed via ‘sh (1)’. To
read a file whose name begins with a bang, prefix the name with a
backslash (‘\’). The default filename is set to FILE only if it is not
prefixed with a bang.
```

1.4 ed.guide/Line addressing

```
Line addressing
***************

   An address represents the number of a line in the buffer. ‘ed’
```

maintains a "current address" which is typically supplied to commands as the default address when none is specified. When a file is first read, the current address is set to the last line of the file. In general, the current address is set to the last line affected by a command.

A line address is constructed from one of the bases in the list below, optionally followed by a numeric offset. The offset may include any combination of digits, operators (i.e., ‘+’, ‘-’ and ‘^’) and whitespace. Addresses are read from left to right, and their values are computed relative to the current address.

One exception to the rule that addresses represent line numbers is the address ‘0’ (zero). This means "before the first line," and is legal wherever it makes sense.

An address range is two addresses separated either by a comma or semicolon. The value of the first address in a range cannot exceed the value of the the second. If only one address is given in a range, then the second address is set to the given address. If an N-tuple of addresses is given where N > 2, then the corresponding range is determined by the last two addresses in the N-tuple. If only one address is expected, then the last address is used.

Each address in a comma-delimited range is interpreted relative to the current address. In a semicolon-delimited range, the first address is used to set the current address, and the second address is interpreted relative to the first.

The following address symbols are recognized.

‘.’
: The current line (address) in the buffer.

‘$’
: The last line in the buffer.

‘N’
: The Nth, line in the buffer where N is a number in the range ‘0,$’.

‘-’
‘^’
: The previous line. This is equivalent to ‘-1’ and may be repeated with cumulative effect.

‘-N’
‘^N’
: The Nth previous line, where N is a non-negative number.

‘+’
: The next line. This is equivalent to ‘+1’ and may be repeated with cumulative effect.

‘+N’
‘WHITESPACE N’
: The Nth next line, where N is a non-negative number. Whitespace followed by a number N is interpreted as ‘+N’.

‘,’
‘%’
     The first through last lines in the buffer. This is equivalent to
     the address range ‘1,$’.

‘;’
     The current through last lines in the buffer. This is equivalent
     to the address range ‘.,$’.

‘/RE/’
     The next line containing the regular expression RE. The search
     wraps to the beginning of the buffer and continues down to the
     current line, if necessary. ‘//’ repeats the last search.

‘?RE?’
     The previous line containing the regular expression RE. The
     search wraps to the end of the buffer and continues up to the
     current line, if necessary. ‘??’ repeats the last search.

‘’LC’
     The line previously marked by a ‘k’ (mark) command, where LC is a
     lower case letter.

1.5 ed.guide/Regular expressions

Regular expressions
****

   Regular expressions are patterns used in selecting text. For
example, the ‘ed’ command

     g/STRING/

prints all lines containing STRING. Regular expressions are also used
by the ‘s’ command for selecting old text to be replaced with new.

   In addition to a specifying string literals, regular expressions can
represent classes of strings. Strings thus represented are said to be
matched by the corresponding regular expression. If it is possible for
a regular expression to match several strings in a line, then the
left-most longest match is the one selected.

   The following symbols are used in constructing regular expressions:

‘C’
     Any character C not listed below, including ‘{’, ‘}’, ‘(’, ‘)’,
     ‘<’ and ‘>’, matches itself.

‘\C’
     Any backslash-escaped character C, other than ‘{’, ‘‘}’, ‘(’, ‘)’,
     ‘<’, ‘>’, ‘b’, ‘B’, ‘w’, ‘W’, ‘+’ and ‘?’, matches itself.

‘.’

Matches any single character.

'[CHAR-CLASS]'
Matches any single character in CHAR-CLASS. To include a ']' in CHAR-CLASS, it must be the first character. A range of characters may be specified by separating the end characters of the range with a '-', e.g., 'a-z' specifies the lower case characters. The following literal expressions can also be used in CHAR-CLASS to specify sets of characters:

[:alnum:] [:cntrl:] [:lower:] [:space:]
[:alpha:] [:digit:] [:print:] [:upper:]
[:blank:] [:graph:] [:punct:] [:xdigit:]

If '-' appears as the first or last character of CHAR-CLASS, then it matches itself. All other characters in CHAR-CLASS match themselves.

Patterns in CHAR-CLASS of the form:
[.COL-ELM.]
[=COL-ELM=]

where COL-ELM is a "collating element" are interpreted according to 'locale (5)' (not currently supported). See 'regex (3)' for an explanation of these constructs.

'[^CHAR-CLASS]'
Matches any single character, other than newline, not in CHAR-CLASS. CHAR-CLASS is defined as above.

'^'
If '^' is the first character of a regular expression, then it anchors the regular expression to the beginning of a line. Otherwise, it matches itself.

'$'
If '$' is the last character of a regular expression, it anchors the regular expression to the end of a line. Otherwise, it matches itself.

'\(RE\)'
Defines a (possibly null) subexpression RE. Subexpressions may be nested. A subsequent backreference of the form '\N', where N is a number in the range [1,9], expands to the text matched by the Nth subexpression. For example, the regular expression '\(a.c\)\1' matches the string 'abcabc', but not 'abcadc'. Subexpressions are ordered relative to their left delimiter.

'*'
Matches the single character regular expression or subexpression immediately preceding it zero or more times. If '*' is the first character of a regular expression or subexpression, then it matches itself. The '*' operator sometimes yields unexpected results. For example, the regular expression 'b*' matches the beginning of the string 'abbb', as opposed to the substring 'bbb', since a null match is the only left-most match.

‘\{N,M\}’
‘\{N,\}’
‘\{N\}’
Matches the single character regular expression or subexpression immediately preceding it at least N and at most M times. If M is omitted, then it matches at least N times. If the comma is also omitted, then it matches exactly N times. If any of these forms occurs first in a regular expression or subexpression, then it is interpreted literally (i.e., the regular expression ‘\{2\}’ matches the string ‘{2}’, and so on).

‘\<’
‘\>’
Anchors the single character regular expression or subexpression immediately following it to the beginning (in the case of ‘\<’) or ending (in the case of ‘\>’) of a "word", i.e., in ASCII, a maximal string of alphanumeric characters, including the underscore (_).

The following extended operators are preceded by a backslash ‘\’ to distinguish them from traditional ‘ed’ syntax.

‘\‘’
‘\’’
Unconditionally matches the beginning ‘\‘’ or ending ‘\’’ of a line.

‘\?’
Optionally matches the single character regular expression or subexpression immediately preceding it. For example, the regular expression ‘a[bd]\?c’ matches the strings ‘abc’, ‘adc’ and ‘ac’. If ‘\?’ occurs at the beginning of a regular expressions or subexpression, then it matches a literal ‘?’.

‘\+’
Matches the single character regular expression or subexpression immediately preceding it one or more times. So the regular expression ‘a+’ is shorthand for ‘aa*’. If ‘\+’ occurs at the beginning of a regular expression or subexpression, then it matches a literal ‘+’.

‘\b’
Matches the beginning or ending (null string) of a word. Thus the regular expression ‘\bhello\b’ is equivalent to ‘\<hello\>’. However, ‘\b\b’ is a valid regular expression whereas ‘\<\>’ is not.

‘\B’
Matches (a null string) inside a word.

‘\w’
Matches any character in a word.

‘\W’
Matches any character not in a word.

1.6 ed.guide/Commands

Commands
****

All ‘ed’ commands are single characters, though some require additonal parameters. If a command’s parameters extend over several lines, then each line except for the last must be terminated with a backslash (‘\’).

In general, at most one command is allowed per line. However, most commands accept a print suffix, which is any of ‘p’ (print), ‘l’ (list), or ‘n’ (enumerate), to print the last line affected by the command.

An interrupt (typically ^C) has the effect of aborting the current command and returning the editor to command mode.

‘ed’ recognizes the following commands. The commands are shown together with the default address or address range supplied if none is specified (in parenthesis).

‘(.)a’
: Appends text to the buffer after the addressed line, which may be the address ‘0’ (zero). Text is entered in input mode. The current address is set to last line entered.

‘(.,.)c’
: Changes lines in the buffer. The addressed lines are deleted from the buffer, and text is appended in their place. Text is entered in input mode. The current address is set to last line entered.

‘(.,.)d’
: Deletes the addressed lines from the buffer. If there is a line after the deleted range, then the current address is set to this line. Otherwise the current address is set to the line before the deleted range.

‘e FILE’
: Edits FILE, and sets the default filename. If FILE is not specified, then the default filename is used. Any lines in the buffer are deleted before the new file is read. The current address is set to the last line read.

‘e !COMMAND’
: Edits the standard output of ‘!COMMAND’, (see the ‘!’ command below). The default filename is unchanged. Any lines in the buffer are deleted before the output of COMMAND is read. The current address is set to the last line read.

‘E FILE’
: Edits FILE unconditionally. This is similar to the ‘e’ command, except that unwritten changes are discarded without warning. The current address is set to the last line read.

‘f FILE’

Sets the default filename to FILE. If FILE is not specified, then the default unescaped filename is printed.

'(1,$)g /RE/COMMAND-LIST'
Applies COMMAND-LIST to each of the addressed lines matching a regular expression RE. The current address is set to the line currently matched before COMMAND-LIST is executed. At the end of the 'g' command, the current address is set to the last line affected by COMMAND-LIST.

Each command in COMMAND-LIST must be on a separate line, and every line except for the last must be terminated by a backslash ('\'). Any commands are allowed, except for 'g', 'G', 'v', and 'V'. By default, a newline alone in COMMAND-LIST is equivalent to a 'p' command. If 'ed' is invoked with the command-line option '-G', then a newline in COMMAND-LIST is equivalent to a '.+1p' command.

'(1,$)G /RE/'
Interactively edits the addressed lines matching a regular expression RE. For each matching line, the line is printed, the current address is set, and the user is prompted to enter a COMMAND-LIST. At the end of the 'G' command, the current address is set to the last line affected by (the last) COMMAND-LIST.

The format of COMMAND-LIST is the same as that of the 'g' command. A newline alone acts as a null command list. A single '&' repeats the last non-null command list.

'H'
Toggles the printing of error explanations. By default, explanations are not printed. It is recommended that ed scripts begin with this command to aid in debugging.

'h'
Prints an explanation of the last error.

'(.)i'
Inserts text in the buffer before the current line. Text is entered in input mode. The current address is set to the last line entered.

'(.,.+1)j'
Joins the addressed lines. The addressed lines are deleted from the buffer and replaced by a single line containing their joined text. The current address is set to the resultant line.

'(.)k LC'
Marks a line with a lower case letter LC. The line can then be addressed as ''LC' (i.e., a single quote followed by LC) in subsequent commands. The mark is not cleared until the line is deleted or otherwise modified.

'(.,.)l'
Prints the addressed lines unambiguously. If invoked from a terminal, 'ed' pauses at the end of each page until a newline is entered. The current address is set to the last line printed.

'(.,.)m(.)'
Moves lines in the buffer. The addressed lines are moved to after the right-hand destination address, which may be the address '0' (zero). The current address is set to the last line moved.

'(.,.)n'
Prints the addressed lines along with their line numbers. The current address is set to the last line printed.

'(.,.)p'
Prints the addressed lines. If invoked from a terminal, 'ed' pauses at the end of each page until a newline is entered. The current address is set to the last line printed.

'P'
Toggles the command prompt on and off. Unless a prompt is specified with command-line option '-p STRING', the command prompt is by default turned off.

'q'
Quits 'ed'.

'Q'
Quits 'ed' unconditionally. This is similar to the 'q' command, except that unwritten changes are discarded without warning.

'($)r FILE'
Reads FILE to after the addressed line. If FILE is not specified, then the default filename is used. If there is no default filename prior to the command, then the default filename is set to FILE. Otherwise, the default filename is unchanged. The current address is set to the last line read.

'($)r !COMMAND'
Reads to after the addressed line the standard output of '!command', (see the '!' command below). The default filename is unchanged. The current address is set to the last line read.

'(.,.)s /RE/REPLACEMENT/'
'(.,.)s /RE/REPLACEMENT/g'
'(.,.)s /RE/REPLACEMENT/n'
Replaces text in the addressed lines matching a regular expression RE with REPLACEMENT. By default, only the first match in each line is replaced. If the 'g' (global) suffix is given, then every match to be replaced. The 'n' suffix, where N is a postive number, causes only the Nth match to be replaced. It is an error if no substitutions are performed on any of the addressed lines. The current address is set the last line affected.

RE and REPLACEMENT may be delimited by any character other than space and newline (see the 's' command below). If one or two of the last delimiters is omitted, then the last line affected is printed as though the print suffix 'p' were specified.

An unescaped '&' in REPLACEMENT is replaced by the currently matched text. The character sequence '\M' where M is a number in the range [1,9], is replaced by the Mth backreference expression

of the matched text. If REPLACEMENT consists of a single '%', then REPLACEMENT from the last substitution is used. Newlines may be embedded in REPLACEMENT if they are escaped with a backslash ('\').

'(.,.)s'
Repeats the last substitution. This form of the 's' command accepts a count suffix N, or any combination of the characters 'r', 'g', and 'p'. If a count suffix N is given, then only the Nth match is replaced. The 'r' suffix causes the regular expression of the last search to be used instead of the that of the last substitution. The 'g' suffix toggles the global suffix of the last substitution. The 'p' suffix toggles the print suffix of the last substitution The current address is set to the last line affected.

'(.,.)t(.)'
Copies (i.e., transfers) the addressed lines to after the right-hand destination address, which may be the address '0' (zero). The current address is set to the last line copied.

'u'
Undoes the last command and restores the current address to what it was before the command. The global commands 'g', 'G', 'v', and 'V' are treated as a single command by undo. 'u' is its own inverse.

'(1,$)v /RE/COMMAND-LIST'
Applies COMMAND-LIST to each of the addressed lines not matching a regular expression RE. This is similar to the 'g' command.

'(1,$)V /RE/'
Interactively edits the addressed lines not matching a regular expression RE. This is similar to the 'G' command.

'(1,$)w FILE'
Writes the addressed lines to FILE. Any previous contents of FILE is lost without warning. If there is no default filename, then the default filename is set to FILE, otherwise it is unchanged. If no filename is specified, then the default filename is used. The current address is unchanged.

'(1,$)wq FILE'
Writes the addressed lines to FILE, and then executes a 'q' command.

'(1,$)w !COMMAND'
Writes the addressed lines to the standard input of '!COMMAND', (see the '!' command below). The default filename and current address are unchanged.

'(1,$)W FILE'
Appends the addressed lines to the end of FILE. This is similar to the 'w' command, expect that the previous contents of file is not clobbered. The current address is unchanged.

'(.)x'

```
     Copies (puts) the contents of the cut buffer to after the
     addressed line.  The current address is set to the last line
     copied.

`(.,.)y'
     Copies (yanks) the addressed lines to the cut buffer.  The cut
     buffer is overwritten by subsequent `y', `s', `j', `d', or `c'
     commands.  The current address is unchanged.

`(.+1)z N'
     Scrolls N lines at a time starting at addressed line.  If N is not
     specified, then the current window size is used.  The current
     address is set to the last line printed.

`! COMMAND'
     Executes COMMAND via `sh (1)'.  If the first character of COMMAND
     is `!', then it is replaced by text of the previous `!COMMAND'.
     `ed' does not process COMMAND for backslash (`\') escapes.
     However, an unescaped `%' is replaced by the default filename.
     When the shell returns from execution, a `!' is printed to the
     standard output.  The current line is unchanged.

`(.,.)#'
     Begins a comment;  the rest of the line, up to a newline, is
     ignored.  If a line address followed by a semicolon is given, then
     the current address is set to that address.  Otherwise, the
     current address is unchanged.

`($)='
     Prints the line number of the addressed line.

`(.+1)newline'
     Prints the addressed line, and sets the current address to that
     line.
```

1.7 ed.guide/Limitations

```
Limitations
***********

   The buffer files are kept in `/tmp/ed.*'.  If the terminal hangs up,
`ed' attempts to write the buffer to file `ed.hup'.

   `ed' processes FILE arguments for backslash escapes, i.e., in a
filename, any characters preceded by a backslash (`\') are interpreted
literally.

   If a text (non-binary) file is not terminated by a newline character,
then `ed' appends one on reading/writing it.  In the case of a binary
file, `ed' does not append a newline on reading/writing.

   Per line overhead: 4 `int's.
```

1.8 ed.guide/Diagnostics

```
Diagnostics
***********

   When an error occurs, if ‘ed’’s input is from a regular file or here
document, then it exits, otherwise it prints a ‘?’ and returns to
command mode.  An explanation of the last error can be printed with the
‘h’ (help) command.

If the ‘u’ (undo) command occurs in a global command list, then the
command list is executed only once.

   Attempting to quit ‘ed’ or edit another file before writing a
modified buffer results in an error.  If the command is entered a
second time, it succeeds, but any changes to the buffer are lost.

   ‘ed’ exits with 0 if no errors occurred; otherwise >0.
```