DATE: November 16, 1973

TO: Distribution

FROM: M. A. Padlipsky

SUBJECT: neted: A Common Editor for the ARPA Network

Attached is draft documentation for:

neted: A Common Editor for the ARPA Network

Your comments are solicited.

Mike Padlipsky

USAGE OF NETED, AN ARPA NETWORK COMMON EDITOR

Introduction

As is typical of "context editors", the NETED command is used both for creating new files and for altering already existing files -- where "files" are named collections of character encoded data in the storage hierarchy of a time-sharing system. Consequently, NETED operates in two distinct "modes" -- called "input mode" and "edit mode".

When NETED is used to create a file (that is, when it is invoked from command level with an argument which specifies the name of a file which does not already exist in the user's "working directory"), it is automatically in input mode. It will announce this fact by outputting a message along the lines of "File soandso not found. Input." Until you take explicit action to leave input mode, everything you type will go into the specified file. (Actually, it goes into a "working copy" of the file, and into the real file only when you indicate a desire to have that happen.) To leave input mode, type a line consisting of only a period and the appropriate new-line character: ".<NL>", where <NL> is whatever it takes to cause a Telnet New-Line to be generated from your terminal.

After leaving input mode, you are in edit mode. Here, you may issue various "requests" which will allow you to alter the contents of the (working) file, re-enter input mode if you wish, and eventually cause the file to be stored. Note that edit mode is entered automatically if the argument you supplied to NETED specified an existing file. Regardless of how it was entered, being in edit mode is confirmed by NETED's outputting a message of the form "Edit." Editing is performed relative to a (conceptual) pointer which specifies the current line, and many requests pertain to either moving the pointer or changing the contents of the current line. (When edit mode is entered from input mode, the pointer is at the last line input; when entered from command level, the pointer is at the "top" of the file.)

Requests

NETED'S edit mode requests follow, in an order intended to be helpful. Two important reminders: the requests may only be issued from edit mode, and each one "is a line" (i.e., terminates in a newline / carriage return / linefeed as appropriate to the User Telnet being employed). <u>Syntax</u> <u>Note</u>: If the request takes an argument, there must be at least one space (blank) between the request's name and the argument.

1. n <u>m</u>

For unsigned <u>m</u>, the n(ext) request causes the pointer to be moved "down" <u>m</u> lines. If <u>m</u> is negative, the pointer is moved "up" <u>m</u> lines. If <u>m</u> is not specified, the pointer is moved one line. If the end of the file is reached, an "End of file reached by n <u>m</u>" message is output by NETED; the pointer is left "after" the last line.

2. l <u>string</u>

The l(ocate) request causes the pointer to be moved to the next line containing the character string <u>string</u> (which may contain blanks); the line is output. If no match is found, a message of the form "End of file reached by l <u>string</u>" will be output (and the pointer will have returned to the top of the file). The search will not wrap around the end of the file; however, if the string was above the starting position of the pointer, a repetition of the locate request will find it, in view of the fact that the pointer would have been moved to the top of the file. To find any occurrence of the string -- rather than the next occurrence -- it is necessary to move the pointer to the top of the file before doing the locate (see following request).

3. t

Move the pointer to the top of the file.

4. b

Move the pointer to the bottom of the file and enter input mode.

5. .

Leave the pointer where it is and enter input mode. (First new line goes after current old line.)

6. i <u>string</u>

The i(nsert) request casues a line consisting of <u>string</u> (which will probably contain blanks) to be inserted after the current line. The pointer is moved to the new line. Edit mode is not left.

7. r <u>string</u>

The r(eplace) request causes a line consisting of <u>string</u> (probably containing blanks) to replace the current line.

8. p <u>m</u>

The p(rint) request casues the current line and the succeding <u>m</u> - 1 lines to be output. If <u>m</u> is not specified, only the current line will be output. End of file considerations are the same as with "n".

9. c /<u>s1</u>/<u>s2</u>/ <u>m</u> g

The c(hange) request is quite powerful, although perhaps a bit complex to new users. In the line being pointed at, the string of characters <u>s1</u> is replaced by the string of characters <u>s2</u>. If <u>s1</u> is void, <u>s2</u> will be inserted at the beginning of the line; if <u>s2</u> is void, <u>s1</u> will be deleted from the line. Any character not appearing within either character string may be used in place of the slash (/) as a delimiter. If a number, <u>m</u>, is present, the request will affect <u>m</u> lines, starting with the one being pointed at. All lines in which a change was made are printed. The pointer is left at the last line scanned. If the letter "g" is absent (after the final delimiter) only the first occurrence of <u>s1</u> within a line will be changed. If "g" (for "global") is present, all occurrences of <u>s1</u> within a line will be changed. (If <u>s1</u> is void, "g" has no effect.) <u>Note well</u>: blanks in both strings are significant and must be counted exactly. End of file considerations are the same as with "n".

10. d <u>m</u>

The d(elete) request causes <u>m</u> lines, including the current one, to be deleted from the working copy of the file. If <u>m</u> is not specified, only the current line is deleted. The pointer is left at a null line above the first undeleted line. End of file considerations are the same as with "n".

11. w

Write out the working copy into the storage hierarchy but remain in NETED. (Useful for those who fear crashes and don't want to lose all the work performed.)

12. save

Write out the working copy into the storage hierarchy and exit from NETED.

<u>Examples</u>

1. Input and Edit modes

Assuming that there is no file named "sample" in your directory, the command

```
          neted sample
```

would cause the response

```
          File not found.
          Input.
```

Typing the following

```
          This is line 1.
          This is line 2.
          This is line 3.
          .
```

would cause the three lines of text to be placed in the working copy of the file, and generate the response (because of the mode change request ".")

```
          Edit.
```

The following sequence would write a copy of the working copy out, move the conceptual pointer to the top of the file, insert a line there, then re-enter Input mode at the bottom of the file:

```
          w
          t
          i This is line 0.
          b
```

(Response after the "b" request is "Input.") Now we add two lines at the bottom and return to Edit mode:

```
          This is line 4.
          This is line 5.
          .
```

(Response is "Edit.") At this point,

```
          save
```

will write out the (six-line) file and return to command level. Note that had it been desired to input more than one line at the top of the file (or elsewhere in the file) the "." request could have been used conveniently to enter Input mode.

2. Pointer-moving requests

Continuing with the file "sample", the following would leave the pointer at the final line:

```
          neted sample
          Edit.                   (response)
          n 6
```

Note that the argument to the "n" request is "6" rather than "5" because the top of the file is a null line rather than the first line. (If you had done an immediate "p" request after entering Edit mode from command level, the response would have been "No line.") An alternative way of moving the pointer to the last line (instead of "n 6") is

```
          l 5
          This is line 5.                (response)
```

This latter method, usually known as "locating by context," is the more common. At this point,

```
          n -2
          p
```

would cause the response

```
          This is line 3.
```

As noted above, "t" moves the pointer to the top of the file, and "b" moves it to the bottom (and enters Input mode).

3. Changing existing lines

Assume the pointer is still located at "This is line 3."

```
          c /is/was/
```

would result in

```
          Thwas is line 3.
```

Ah well. Blanks <u>are</u> significant. To fix the mess and do what was intended:

```
          c /was/is/
          This is line 3.                (response)
          c / is/ was/
          This was line 3.               (response)
```

To change all instances of a character string on a given line:

```
          c /i/x/ g
          Thxs was lxne 3.               (response)
```

(Note the space before the "g".) An easy way to fix that line would be

```
          r This is line 3.
```

which simply replaces the current line. ("c /x/1/ g" would also work, of course.)

The following request (the pointer is not changed by the "r" request)

```
          c /line/entry/ 2 g
```

would result in the response

```
          This is entry 3.
          This is entry 4.
```

with the pointer now at "This is line 4."

To append to the beginning of a line,

```
          c //tag:/
          tag:This is line 4.            (response)
```

And to remove a string from a line,

```
          c /tag://
          This is line 4.                (response)
```

Note that "/" need not be used as the delimiter. I.e., "c xtag:xx" would also have worked in the last instance.

4. Miscellaneous requests

Still using "sample" consider the following:

```
          t
          p
          No line.             (response)
          n
          p
          This is line 0.                (response)
          d 2
          p
          No line.             (response)
          i This is the begining.
          c /in/inn/
          This is the beginning.                   (response)
          l 3
          This is line 3.                (response)
          d 99
          End of file reached by "d 99"            (response)
          .
          Input.               (response)
          This is the end.
          .
```

```
Edit.                   (response)
t
p 99
No line.                (response)
This is the beginning.                      (response)
This is line 2.                   (response)
This is the end.                  (response)
End of file reached by "p 99".              (response)
```

Note that the first "d" request took care of the lines ending with "0." and "1." and the second took care of "3." through "5." The "." after the "d 99" could also have been a "b" or an "i" request. A "save" request at this point would leave you with a file containing only the three text lines which were printed in response to the "p 99".

5. Additional features in certain implementations

Some implementations of NETED will give "prompts" when a new request is expected; the prompt is an asterisk ("*") without a carriage return. Two additional requests may be furnished in some cases: "g"(et) <u>filename</u> (which reads an existing file into the working file) and "quit" (which exits without saving the work performed since the last "w" request). Note: the "quit" function is always performable by means of a Telnet Protocol "Interrupt Process" generic function. Finally, some implementations will offer "self-documentation" of the command, in response to a "?" or "h"(elp) request.